Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«Уральский федеральный университет имени первого Президента России Б. Н. Ельцина»

УТВЕРЖДАЮ

Директор по образовательной деятельности

_______________ С.Т. Князев

«24» декабря _______________ 2021 г.

Компьютерное зрение

Учебно-методические материалы по направлению подготовки
09.04.01 Информатика и вычислительная техника
Образовательная программа «Инженерия искусственного интеллекта»

Екатеринбург

2021

РАЗРАБОТЧИКИ УЧЕБНО-МЕТОДИЧЕСКИХ МАТЕРИАЛОВ

Доцент, канд.техн.наук		Ронкин Михаил Владимирович

СОДЕРЖАНИЕ

Лекция 1

Лекция сопровождается презентацией.

Тема: Современные подходы к задачам компьютерного зрения

Задачи:

- Обзор некоторых задач компьютерного зрения;
- Особенности представления изображения в цифровом виде;
- Принципы цифровой обработки изображений;
- Основные операции цифровой обработки изображений

Лекция является вводной в предмет «Компьютерное зрение». В рамках лекции нужно рассказать о предмете «Компьютерное зрение» и типичных задачах. Также нужно показать, что такое изображение. Часто студенты не понимают, что изображение — это набор матриц, им нужно это показать. Также можно упомянуть термин «Тензор», и сказать, что в случае обработки изображений — это просто набор матриц. Более того нужно сказать о том, что изображения могут быть многоканальными и одноканальными. Нужно сказать о том, что такое канал и значения пикселей в нем. После можно рассказать об особенностях цифровых изображений. В том числе, о том, что такое дискретизация, проблема шумов и т.д. В целом про все эти вещи нужно рассказывать обзорно. Важно, чтобы слушатели поняли сам принцип и не так важно понять суть.

После рассказа о цифровом изображении нужно сказать о том, что мы можем обрабатывать изображения на разном уровне. Мы можем поставить задачу удаления шумов, а можем поставить задачу формирования отчета о состоянии здоровья по снимкам. Первую задачу мы можем решить классическими методами, но для второй нам потребуется глубокое обучение. Можно привести и другие примеры задач где нужны те или иные подходы. При этом нужно отметить, что мы используем глубокое обучение там, где нет других подходов. Однако, в компьютерном зрении это частая ситуация, так как сложно формализовать математическую модель задачи. Например, сложно (на сегодня не возможно) описать лицо человека во всех вариантах его изображения, но глубоким обучением решить задачу распознавания лица очень просто.

В данной лекции предполагается, что будут показаны примеры классических техник обработки изображений – как самой простой задачи компьютерного зрения. Среди подходов следует уделить наибольшее внимание операции свертка. Это операцию надо пояснить подробно. Лучше сопроводить это комментарием о том, что операция будет использоваться во всем курсе, так как в основном мы используем глубокие сверточные нейронные сети (большинство задач компьютерного зрения решаются сегодня ими).

В конце лекции, если будет время, можно показать и разобрать пример задачи компьютерного зрения, решаемой классическим методами. Пример это метод повышения качества рентгеновского изображения человека. Тут нужно отметить уровень задачи – это именно повышение качества, а не принятие решения о состояние здоровья или не задача выделения наиболее подозрительных (на определенные болезни) участков и т.д.

Отметим, что вторая часть данной лекции (классические подходы), а также лекция 2, могут быть прочитаны в конце курса. В этом случае начать стоит с лекции 5 – сверточная нейронная сеть. Также можно поступить таким образом, что прочитать сжато вторую часть данной лекции и лекцию 2 за одну лекцию. В этом случае будет полезным разбить на две лекции или 6 лекцию (архитектуры) или 8 (остальное). Данный подход также позволяет начать с 4 лекции. То есть в таком порядке: 1-вводная часть, 5,3,4, 6 часть 1, 6 часть 2, 7,8, 1-вторая часть, 2 лекции. Или: 1-вводная часть, 5,3,4, 6,7,8-1 часть, 8-2 часть, 1-вторая часть, 2 лекции.

В случае, если выбран порядок, когда после 1 части данной лекции читается что-то кроме ее 2й части и лекции 2 потом, то полезно будет пояснить понятие признак для изображений. Важно отметить важность этого понятия при решение высокоуровневых задач компьютерного зрения. Также полезно привести примеры разного рода признаков.

Лекция 2

Лекция сопровождается презентацией.

Тема: Особенности использования методов машинного обучения в задачах компьютерного зрения

- **Задачи**:
    - Предмет машинного обучения;
    - Виды признаков изображений;
    - Обзор некоторых методов решения задач компьютерного зрения с использованием машинного обучение;
    - Особенности глубоких нейронных сетей и их место среди методов решения задач компьютерного зрения;
    - Метод градиентного спуска;
    - Метод обратного распространения ошибки.

В данной лекции необходимо объяснить слушателям цель использования машинного обучения при решении задач компьютерного зрения. Можно начать с приведенных примеров, можно привести какие-то еще примеры использования классических подходов машинного обучения. Например, можно упомянуть о том, что до волны глубокого обучения наиболее популярными были подходы с использованием метода опорных векторов в задачах классификации и многих других. При этом вектора могли работать как над самим изображением, преобразованным в вектор, так и над рядом характеристик, полученных из изображения.

Одним из самых важных понятий лекции является понятие признак. Нужно объяснить, что значит признак для изображения. Как правило этот термин вызывает ряд вопросов у слушателей, так как является достаточно широким. Поэтому лучше остановиться на этой теме подробней.

В лекции приведены примеры того, как можно получить признаки. Следует пояснить эти примеры слушателям. С данными методами слушатели также ознакомятся в ходе практических работ.

Важно пояснить что сами признаки могут быть полезными, а могу и нет. Важно сказать о том, что такое выборка и зачем она нужна – мы хотим получить регулярные признаки. Причем, мы можем получить их для каких-то групп ответов или просто регулярные. Также мы можем рассмотреть каждый пиксель как признак, признаки могут быть разной информативности (абстрактно).

После разъяснения термина признак следует перейти к тому, что признаки мы обрабатываем с помощью методов машинного обучения. При этом мы можем решить достаточно высокоуровневые задачи. Следует пояснить, что в случае обработки изображений, машинные методы кроме глубокого обучения часто позволяют решать это не самые сложные задачи, зато решаются достаточно быстро. Например, можно сказать о том, что с использованием классических методов можно получить алгоритмы типа «полуавтомат», но сложно получить полностью автоматические алгоритмы. Можно привести несколько примеров использования машинного обучения.

Далее следует разъяснить, что мы можем выделять признаки сами, а можем сделать это автоматически. Так как признаки достаточно сложно формулируются, то самим выделить их бывает очень сложно и затратно. Методы глубокого обучения нейронных сетей позволяют выделять признаки автоматически – это главная причина их популярности в глубоком обучении нейронных сетей.

Далее следует сказать, что такое нейронные сети. Нужно сказать, что это пример нелинейной регрессии. Далее следует кратко пояснить что такое линейная регрессия, перейти к нелинейной регрессии, ее можно пояснить на примере логиситческой регрессии. Далее нужно сказать, что данная регрессия решается методом градиентного спуска. Следует достаточно подробно объяснить суть метода градиентного спуска.

После объяснения метода градиентного спуска нужно сказать о том, что метод позволяет получить хорошие результаты, но что делать, если нелинейных регрессоров (т.е. персептронов) много. Далее нужно показать, что от метода градиентного спуска мы переходим к методу обратного распространения ошибки.

В данной лекции метод обратного распространения ошибки показан на примере полносвязной сети. Важно пояснить этот метод. Далее можно обозначить тот факт, что для сверточной сети метод будет работать по тому же принципу, но вместо отдельных персептронов там будут сверточные ядра с обучаемыми весами. Объяснение метод градиентного спуска для сверточной нейронной сети выходит за рамки данного курса. Можно это объяснить тем, что в большинстве современных фреймворков работы с нейронными сетями этот метод реализован автоматически.

Если остается время, можно рассказать о других видах нейронных сетей и сказать о том, почему именно сверточная сеть используется в данных приложениях.

Отметим, что данная лекция, а также часть лекции 1, посвященная классическим методам могут быть прочитаны в конце курса. В этом случае начать стоит с лекции 5 – сверточная нейронная сеть. Также можно поступить таким образом, что прочитать сжато данную и 1 лекцию вместе. В этом случае будет полезным разбить на две или 6 лекцию (архитектуры) или 8 (остальное).

Лекция 3

Лекция сопровождается презентацией.

Тема: Особенности искусственных нейронных сетей в задачах компьютерного зрения

- **Задачи**:
- Стохастический градиентный спуск и его виды;
- Выбор скорости обучения;
- Проблемы обучения методом обратного распространения ошибки;
- Обзор функций активации;
- Инициализация весовых параметров нейронных сетей.

Данная лекция посвящена вопросам обучения нейронных сетей. В лекции нужно напомнить о том, что такое градиентный спуск и метод обратного распространения ошибки. Далее нужно сказать о том, что данных у нас очень много и о том, почему мы используем стохастический градиентный спуск. Далее нужно сказать о том, что выбор скорости обучения — это проблемы. Проблемы мы можем решить путем варьирования скорости обучения или путем использования адаптивных методов градиентного спуска. Также можно сказать о том, что есть методы оптимизации, не требующие выбора данного параметра, но как правила у этих методов другие недостатки, например, требования к рельефу функции потерь или высокая вычислительная сложность, проще выбрать скорость обучения. Также тут можно упомянуть о проблемах градиентного спуска – попадание в локальные минимумы, переобучение, вымывание или взрыв градиента. Также тут можно сказать о том, что такое тренировочная, тестовая и валидационная выборки и что такое кросс-валидация.

Во второй части лекции рассматриваются функции активации. Тут важно отметить что такое sigmoid, softmax и ReLU. Также можно указать на их достоинства и недостатки и сказать об их альтернативах.

В третьей части лекции рассматриваются метод инициализации весовых функций. Тут важно сказать о том, что нужно инициализировать параметры и почему это нужно делать. Также важно сказать о том, почему лучше предобучать сеть, чем использовать инициализацию. Также можно упомянуть о том, что такое перенос обучения. Можно сказать, о том, что можно замораживать веса и чем отличается часть backbone и head у сети. Данные вопросы также будут рассмотрены в практической работе.

Лекция 4

Лекция сопровождается презентацией.

Тема: Особенности использования методов машинного обучения в задачах компьютерного зрения

- **Задачи**:
    - особенности выбора функций потерь нейронных сетей;
    - регуляризация обучения нейронных сетей: лассо, Тихонов, дропаут, батчнорм (и др. нормализации);
    - аугментация изображений.

Первая часть данной лекции посвящена выбору функций потерь в глубоких нейронных сетях. В этой части лекции следует сделать акцент на том, что такое бинарная, многоклассовая и многометочная задачи классификации. Также важно объяснить проблемы несбалансированных классов. Можно привести такие примеры: в задачи сегментации большая часть изображения – это фон, бывают, а и очень маленькие объекты – каждый класс не сбалансирован, по общему содержанию (общей площади) в наборе данных. Точно такая же ситуация, например, в обнаружении и локализации объектов, хотя и не такая интуитивная. Поэтому в компьютерном зрении важно использовать функции потерь для несбалансированных классов. Также важно объяснить, что функции потерь могут быть линейными комбинациями разных функционалов. Таким образом удается решать задачи одновременной оптимизации. Примерами таких задач является обнаружение и локализация объектов, когда мы хотим одновременно и классифицировать объект и сказать где он на изображении. Также в данной части можно упомянуть о задаче семантической сегментации. Для слушателей часто оказывается не понятным тот факт, что семантическая сегментация – это классификация по пикселям. Поэтому лучше этот тут отметить. Также можно сказать о специфических видах функций потерь в обнаружении объектов и сегментации, в том числе упомянуть что такое IoU, Dice, и т.д.

Вторая часть лекции посвящена вопросам регуляризации. Тут нужно рассказать о нескольких вещах. Примеры аугментации, также важно объяснить, что аугментацию нужно использовать аккуратно, иначе можно внести дисбаланс в регулярность и распределения признаков в данных. Кратко можно рассказать об L1/L2 регуляризации. Следует уточнить что такое дропаут и особенно 2D дропаут, в изоображениях нужен именно такой (по-канальный дропаут, так как часто пиксели сильно коррелированы, и выключить одни из них – это ничего не даст). Также можно тут сказать о стохастической глубине, как методе дропаута. Самое важно в этой части лекции — это метод батч-нормализации. Про этот метод нужно объяснить подробно. Нужно также сказать о разнице в работе батч-нормализации на этапах тренировки и работы сети. Также нужно указать достоинства и недостатки метода. После объяснения данного метода следует рассказать об альтернативных подходах и почему они используются.

Лекция 5

Лекция сопровождается презентацией.

Тема: Особенности операций в сверточных нейронных сетях

- **Задачи**:
  - Определение сверточной нейронной сети для задачи классификации,
  - Определение свертки,
  - Типы сверток в сверточных нейронных сетях,
  - Типы операций передескретизации,
  - Операция интерполяции.

Первая часть данной лекции посвящена объяснению того, что такое операция свертки в сверточных нейронных сетях. В том числе, рассказывается о том, что такое сама сверточная сеть, ее достоинства и недостатки. Тут важно объяснить почему для изображений сверточные сети работают. Также важно объяснить концепцию рецептивного поля и концепцию карты признаков.

Касательно операции свертки важно пояснить что эта операция значит для тензоров, что свертки — это трехмерные объекты и их может быть много для получения карт признаков. Также нужно сказать, что такое ядро свертки, зачем нужен паддинг, как связан размер ядра и рецептивное поле и т.д. Если есть время можно сказать о том, как операция свертки реализуется, но это не самое важное для курса.

Далее важно объяснить путь развития идей свертки, почему используют пространственно-разделенную свертку, точечную и глубокую свертки. Также тут можно пояснить что такое операция внимания для сверточных сетей. Также можно рассказать о транспонированной свертке и о свертки повышения разрешения.

Вторая часть лекции посвящена операции пулинга. Тут нужно пояснить что такое ядро пулинга, как оно работает. Важно объяснить, что такое макс-пулинг и глобальный усредняющий пулинг. Про остальные типы можно рассказать бегло.

В конце лекции нужно кратко объяснить, что такое интерполяция, и чем эта операция отличается от пулинга. Следует указать на важность и популярность билинейной интерполяции. Важно отметить, что данная операция может использоваться как для повышения разрешения, так и для его понижения.

Можно также отметить, что в целом операции обратной свертки и свертки высокого разрешения являются альтернативными для обратного пулинга и интерполяции в ряде задач. Также можно отметить, что в целом операция свертки с шагом больше 1 эквивалентна макс пулингу, но экономит вычислительные ресурсы. Если есть время, то можно еще отметить, другие виды пулинга, например, пространственный пирамидальный пулинг или RoI пулинг.

Отметим, что данная лекция может быть прочитана в начале курса после введения.

Читая первую часть данной лекции можно также приводить примеры архитектур, в которых используются те или иные виды сверток в их историческом контексте.

Также можно отметить, что не всегда свертка с меньшим числом параметров оказывается выгодней с точки зрения архитектуры вычислительного устройства. В том числе в сетях для мобильных устройств часто отказываются от глубоких пространственных сверток в силу простых вариантов.

Также важно отметить, что чем больше каналов и слоев, тем больше функций активации, то есть нелинейностей. Тогда больше вероятность, что сеть выделит какой-то сложный нелинейный и абстрактный признак.

Лекция 6

Лекция сопровождается презентацией.

Тема: Архитектуры сверточных нейронных сетей в задачах классификации.

- **Задачи**:
    - Обзор архитектур сверточных нейронных сетей в задачах классификации в нескольких исторических периодах (1998-2017, 2017-221).

В данной лекции приводится обзор основных архитектур сверточных нейронных сетей. Важно сказать о том, что архитектуры постоянно развиваются и достаточно быстро. В том числе сегодня вопрос о том какой должна быть архитектура для задач компьютерного зрения является снова открытым. То есть нужны ли только свертки, или их нужно с чем-то комбинировать или вообще без них, сегодня это уже не так очевидно. Однако, на практике чаще всего используют классические сверточные нейронные сети. В т.ч. наиболее популярна сеть 2015 года ResNet.

Лекция начинается в LeNet, однако можно сказать, что архитектуры сверточных сетей начались с когнитрона в 70-х, а первая именно сверточная сеть была предложена Лекном в 1989. Однако, современная история начинается именно с LeNet. В этой архитектуре были заложены основные принципы, которые и сейчас используются в глубоком обучении нейронных сетей. Причем реализации LeNet сегодня отличается от той, которая была изначальна. Также можно обратить внимание, на то, что LeNet была изначальна предложена для решения конкретной задачи – распознавание цифр индексов, поэтому с сетью связан набор данных MNIST.

Далее стоит рассказать об AlexNet, а также предпосылках для его появления, в т.ч. использование GPGPU, relu, dropout и т.д., а также о роли набора данных ImageNet и соответствующего соревнования в этом. Также важно отметить ход развития соревнования ImageNet.

Далее можно кратко пояснить архитектуры в плоть до ResNet. Данную архитектуру следует рассмотреть подробно. Важно рассказать, почему работает остаточная связь. Также важно отметить значение архитектуры до настоящего времени. Также можно отметить, что именно ResNet позволило показать точность, выше чем у эксперта для ImageNet. Следует показать развитие идей ResNet, в т.ч. остановиться на Xception и DenseNet.

Далее важно рассказать о развитие сетей для мобильных и низко производительных устройств. Важно отметить MobileNet.

Также важно отметить SENet как использование идеи Attention.

Важно отметить роль методы NAS. Среди данной группы сетей стоит остановиться на архитектурах MobileNet V3 и EfficientNet.

Отметим, что данную лекцию можно поделить на две части, в первой рассказать о сетях до 2017 года, во второй о более поздних и современных разработках. В т.ч. если остается время можно тут рассказать о 1-й части 8-й лекции – тренды, в т.ч. рассказать о трансформирах и миксерах как альтернативах сверточным сетям.

Очень важно отметить, почему тут рассматриваются именно сети для классификации. Тут следует ввести термин backbone и нужно сказать о том, что backbone. Сетей классификации как правило используется в остальных задачах компьютерного зрения

Лекция 7

Лекция сопровождается презентацией.

Тема: Архитектуры сверточных нейронных сетей в задачах обнаружения объектов.

- **Задачи**:
  - Обзор особенностей архитектур нейронных сетей многоэтапного поиска и выделения объектов на изображениях;
  - Обзор особенностей архитектур одноэтапного поиска и выделения объектов.

В данной лекции приводится краткий обзор архитектур для решения задач поиска и локализации объектов. В лекции важно разъяснить принципы работы таких сетей, как Faster-RCNN, Retina, FPN и YOLO. В первой части лекции нужно разъяснить что такое многоэтапные сети обнаружения объектов, в чем их недостатки и достоинства. Важно рассказать о том, что такое Region Proposal Network, RoI Pooling, Non-Maximum Supression и другие термины. Также можно рассказать о метриках в обнаружении объектов, а также о том, что такое IOU. Важно уточнить особенности Faster-RCNN, эта сеть будет в практической работе. Также можно упомянуть о Mask-RCNN и о том, что в ней значит RoI Align.

Во второй части лекции важно рассказать о развитии идей YOLO. Важно уточнить особенности YOLO 5, эта сеть будет в практической работе. Также важно сказать о практической значимости данной сети.

В ходе лекции следует достаточно подробно объяснить, как работает одноэтапная детекция, в т.ч. о том, как получаются анхоры (анкеры) и как интерпретируются результирующие карты признаков. Эти вопросы вызывают сложности у слушателей.

Если в конце лекции остается время можно рассказать о экземплярной сегментации (instance segmentation). Тут самое важное рассказать о Mask-RCNN. Также следует упомянуть подход YOLACT. Также можно упомянуть об идеи Panoptic Segmentation, как комбинации instance segmentation и semantic segmentation.

Также в ходе лекции важно сказать о главных проблемах object detection. Такими проблемами являются дисбаланс классов (большая часть анхоров пустые), проблема выделения небольших объектов, проблема hard negative mining.

Лекция 8

Лекция сопровождается презентацией.

Тема: Тенденции и другие задачи компьютерного зрения. Семантическая сегментация, экземплярная (объектная) сегментация; генеративные подходы.

- **Задачи**:
  - Тенденции развития нейронных сетей;
  - Семантическая сегментация, экземплярная (объектная) сегментация;
  - генеративные подходы.

В данной лекции приводятся несколько тем сразу. Данные темы не вошли в другие лекции. Все темы могут быть рассмотрены достаточно обзорно. Однако лучше всего подробно рассказать о семантической сегментации.

В первой части лекции следует указать на основные тенденции развития нейронных сетей в настоящее время. В том числе следует рассказать о том, что такое архитектуры трансформеры, почему к ним такой интерес. Важно пояснить, что эти архитектуры в основном сегодня исследуются. Дело в высокой вычислительной сложности. Однако, за счет перемешивания частей изображений тут можно лучше справляться с выделением сложных крупных признаков. Хотя, тут есть проблемы с небольшими локализованными признаками. В связи с этим также есть тенденция на исследование гибридных архитектур и архитектур миксеров.

Также важно сказать о развитии методов обучения нейронных сетей с учителем и учеником, о методах регуляризации и о методах аугментации. Важно объяснить эти методы работают и в чем их достоинства.

Вторая часть лекции посвящена обзору идей экземплярной сегментации. Тут важно рассказать о Mask-RCNN. Эта сеть наиболее популярна в данной области. Также стоит упомянуть о быстрых подходах, подобных YOLACT.

В третьей части лекции важно более подробно рассказать о задачи семантической сегментации. Тут следует рассказать о ходе развития данной идеи. Особенно важно пояснить работу сетей U-Net. Важно также еще раз напомнить, что семантическая сегментация – это задача по-пиксельной классификации. Также можно указать на основные проблемы, в том числе дисбаланс классов.

В последней части лекции рассматриваются генеративные подходы. Тут достаточно просто обзора. Следует сказать о простом подходе GAN и подходе VAE. Эти подходы работают без учителя. Также важно сказать, почему нужен учитель – что сети сложно учить и они часто переобучаются. По данным подходам предусмотрена практическая работа.

Отметим, что данная лекция может быть разбита на две, в случае необходимости. Также можно разбить данную лекцию + 6 и 7 лекцию, сделав из трех лекций 4. Таким образом в 6 лекция будет разбита на две части, в одну из которых включены тенденции и семантическая сегментация; 7 лекция будет разбита на две части, в одну из которых включены экземплярная сегментация и генеративные подходы.

Лекция является завершающей в курсе.

Лабораторная 1

Работа сопровождается документом в формате ipnb

Изучение представлений изображений и классических методов их обработки.

Изучение представлений изображений и классических методов их обработки. Знакомство с библиотекой opencv или skimage. Представление изображения, генерация изображения. Добавления шумов к изображению. Гистограмма яркости изображения. Методы работы с гистограммой яркости. Методы работ с фильтрами изображений.

```
import numpy as np
import matplotlib.pyplot as plt
```

Pillow Library

Библиотека Python Pillow построена на основе PIL (Python Image Library). Библиотека Pillow поддерживает множество форматов файлов изображений, включая BMP, PNG, JPEG и TIFF. Самый важный класс в библиотеке изображений Python - это класс Image, определенный в одноименном модуле. Модуль Pillow Image предоставляет функции open () и show () для чтения и отображения изображения соответственно. Для отображения изображения Pillow сначала преобразует изображение в формат .png (в ОС Windows) и сохраняет его во временном буфере, а затем отображает его. Следовательно, из-за преобразования формата изображения в ".png" некоторые свойства исходного формата файла изображения могут быть потеряны (например, анимация). Поэтому рекомендуется использовать этот метод только в тестовых целях. Кроме того, мы будем использовать библиотеку urllib для открытия изображений из Интернета.

```
import urllib.request
from PIL import Image

image_url =
'https://as2.ftcdn.net/v2/jpg/02/31/48/03/1000_F_231480357_TGpMz4r5HSFAlm43FkZ366FjFZuuoRA8.jpg'

urllib.request.urlretrieve(image_url,"image.png")

img = Image.open("image.png")

img.show();
```

Для изображения из Pillow можно получить несколько характеристик. Атрибут *format* определяет источник изображения. Если изображение не было прочитано из файла, устанавливается значение «None». Атрибут *size* - это кортеж из двух элементов, содержащий ширину и высоту (в пикселях). Атрибут *mode* определяет тип изображений. Общие режимы: 'L' (яркость) для изображений в оттенках серого, "RGB" для полноцветных изображений, "CMYK" для дополненных изображений.

```
print(img.size, img.format, img.mode)
 (1000, 667) JPEG RGB
```

PIL позволяет конвертировать изображения с помощью метода *convert*.
Небольшая хитрость для jupyter: если вы не хотите открывать изображение в новом окне, вы можете использовать имя изображения без методов, как показано ниже

```
img
```

```
img_gray = img.convert('L')
```

```
(1000, 667) None L
```

На самом деле, помимо функции работы с изоображниями как таковыми, библиотека включает ряд преобразований, которые вы можете выполнить с помощью PIL. Например, давайте уменьшим размер

```
size = (512, 512)
img.thumbnail(size)
print(img.size, img.format, img.mode)
img
```

```
(512, 341) JPEG RGB
```

Кроме того, PIL позволяет производить преобразование изображений. Ниже вы можете найти несколько примеров таких операций

- Примечание *, если вы хотите обрезать изображение, лучше взять новое имя в качестве вывода, например *img_new = img.copy ()* для создания новой копии изображения.

обрезка

```
box = (100, 100, 300, 300)
region = img.crop(box)
region
```

поворот

```
img_45 = img.rotate(angle=45)
img_45
```

Траспонирование

```
out = img.transpose(Image.ROTATE_270)
out
```

Переворот

```
out = img.transpose(Image.FLIP_LEFT_RIGHT)
out
```

Перемешивание каналов

```
r, g, b = img.split()
img_brg = Image.merge("RGB", (b, g, r))
img_brg
```

Комбинации методов

```
box = (0, 0, 400, 300)
region = img.crop(box)
region = region.transpose(Image.ROTATE_180)
imgm = img.copy()
imgm.paste(region, box)
imgm
```

Создание нового изображения.

```
new_im = Image.new(mode='RGB', size = (2*img.size[0],1*img.size[1]),
color=(250,250,250))
new_im.paste(img, (0,0))
new_im.paste(imgm, (img.size[0],0))
new_im
```

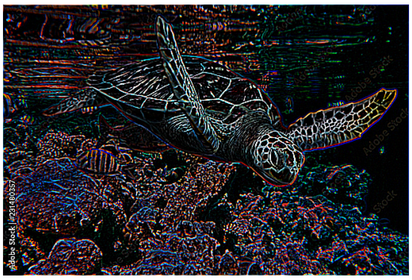
Добавлеие элементов векторонй графики и текста *ImageFont* и `ImageDraw'

```
from PIL import ImageFont
from PIL import ImageDraw

imgt = img.copy()
imgt.resize((512,512))
#line shape
shape = [(490, 300), (10, 10)]

img1 = ImageDraw.Draw(imgt) # create line image
img1.line(shape, width=10)
draw = ImageDraw.Draw(imgt)
draw.text((0, 0), "Hello", (255, 255, 255))
imgt
```

Также PIL содержит несколько предустановленных фильтров улучшения изображения в модулях *ImageFilter* и *ImageEnhance*

```
from PIL import ImageFilter
from PIL import ImageEnhance

out = img.filter(ImageFilter.BLUR)
out
```

```
out = img.filter(ImageFilter.FIND_EDGES)
out
```



```
out = ImageEnhance.Contrast(img)
out = out.enhance(2.3)
out
```

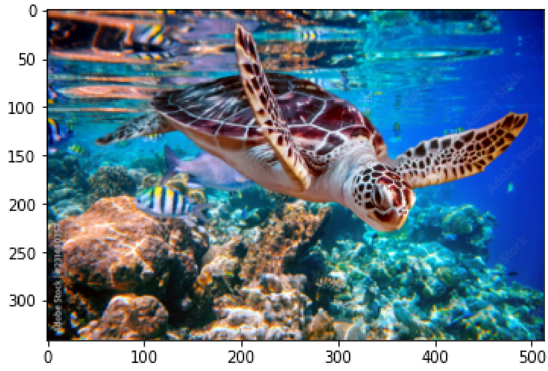
```
out = ImageEnhance.Sharpness(img)
out = out.enhance(23.0)
out
```

Для преобразования изображений из PIL в другие форматы вы можете использовать *np.array ()*.

> *Примечание* для обратного преобразования используйте *Image.fromarray()*. Если вы хотите сохранить изображение, используйте *img.save(path_name)*

```
image = np.array(img)
plt.imshow(image);
print(image.shape)
```

```
(341, 512, 3)
```



Упражнение 1

1. Возьмите несколько некачественных изображений и попробуйте улучшить их качество с помощью Pillow. Например, используйте этот https://thumbs.dreamstime.com/z/abstract-blur-cafe-coffee-shop-abstract-blur-cafe-coffee-shop-background-162453910.jpg.
2. произвести несколько вариантов модификаций (аугментаций) обработанных изображений (минимум 5 вариантов).

Представление цифрового изображения

Проверьте, есть ли у вас *opencv*, либо установите его, используя официальную документацию https://pypi.org/project/opencv-python/

- Вариант 1 - Пакет основных модулей: *pip install opencv-python*
- Вариант 2 - Полный пакет (содержит как основные модули, так и дополнительные модули): *pip install opencv-contrib-python*

```
import cv2 as cv
```

Первое, что нам нужно, это открыть изображение

```
image_url =
'https://as2.ftcdn.net/v2/jpg/02/31/48/03/1000_F_231480357_TGpMz4r5HSFAlm43FkZ366FjFZuuoRA8.jpg'
urllib.request.urlretrieve(image_url,"image.png")

image = cv.imread("image.png")
(h, w, d) = image.shape
print("width={}, height={}, depth={}".format(w, h, d))
```

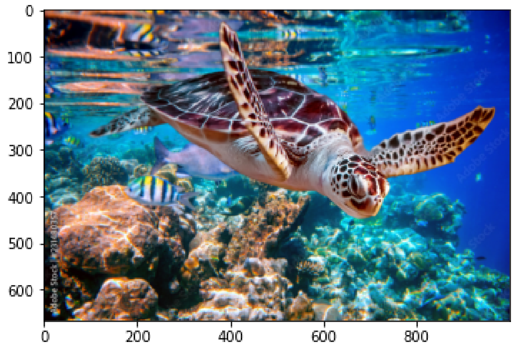
```
width=1000, height=667, depth=3
```

вы можете открыть изображение с помощью встроенного метода

```
cv.imshow("Image", image)
cv.waitKey(0)
cv.destroyAllWindows()
```

Лучше открыть его с помощью *plt* (методы*matplotlib*). Однако первое, что нам нужно сделать при работе с открытыми изображениями *cv* вне библиотеки - это преобразовать формат изображения. Это связано с тем, что *opencv* работает с изображением в формате brg (обратном к rgb).

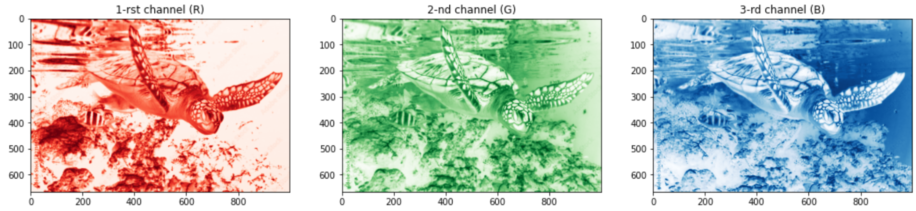
```
image = cv.cvtColor(image, cv.COLOR_BGR2RGB)
plt.imshow(image);
```



Кроме того, было бы полезно посмотреть, как изображения выглядят в матричной форме. Напомним, что обычно для кодирования цифровых изображений используют 8-битный (0-255) формат в трех каналах. Таким образом, каждый канал можно рассматривать как отдельную матрицу.

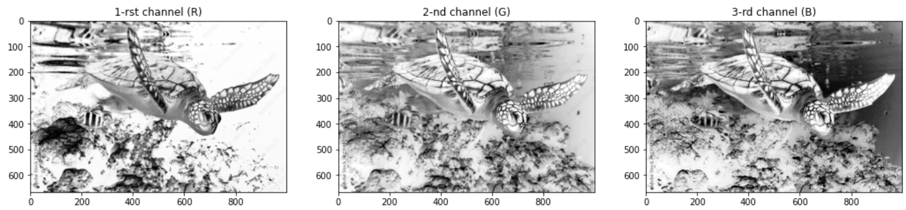
```
print(image.shape)
print(image[:,:,0])
# plt.matshow(image[:,:,0])
# plt.show();
plt.figure(figsize=(18,6));
plt.subplot(1,3,1)
plt.imshow(image[:,:,0],cmap="Reds")
plt.title('1-rst channel (R)')
plt.subplot(1,3,2)
plt.imshow(image[:,:,1],cmap="Greens")
plt.title('2-nd channel (G)')
plt.subplot(1,3,3)
plt.imshow(image[:,:,2],cmap="Blues")
plt.title('3-rd channel (B)')
plt.show();
```

```
(667, 1000, 3)
[[ 83  90 100 ...   1   1   2]
 [ 89  94 105 ...   1   1   2]
 [ 96 101 111 ...   1   1   1]
 ...
 [ 10  10  10 ...  14  13  10]
 [ 10  10  10 ...  15  14  13]
 [ 10  10  10 ...  16  16  15]]
```



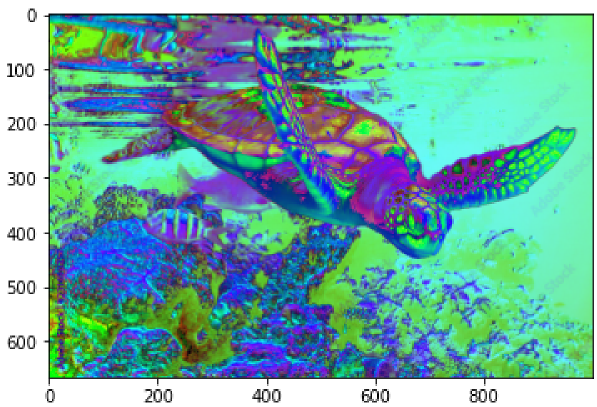
Однако на самом деле было бы лучше отображать каждый канал в оттенках серого, потому что на самом деле цифровое значение - это интенсивность.

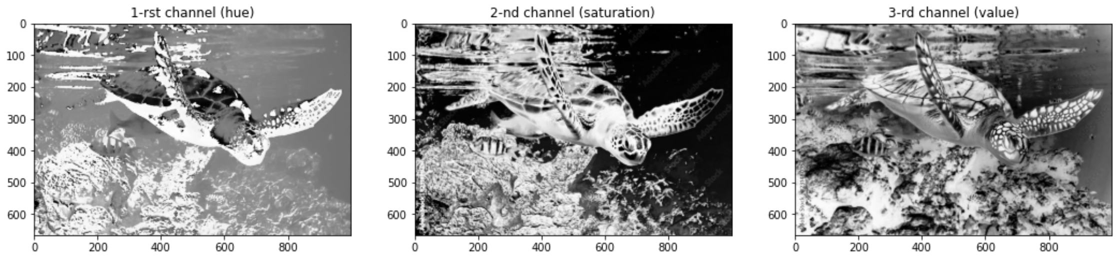
```python
plt.figure(figsize=(18,6));
plt.subplot(1,3,1)
plt.imshow(image[:,:,0],cmap="Greys")
plt.title('1-rst channel (R)')
plt.subplot(1,3,2)
plt.imshow(image[:,:,1],cmap="Greys")
plt.title('2-nd channel (G)')
plt.subplot(1,3,3)
plt.imshow(image[:,:,2],cmap="Greys")
plt.title('3-rd channel (B)')
plt.show();
```



Как вы, наверное, знаете, RGB - это не единственный формат, ниже показан другой вариант - HSV (оттенок, насыщенность, значение интенсивности)

```python
image = cv.imread("image.png")
image=cv.cvtColor(image,cv.COLOR_BGR2HSV)
plt.imshow(image);
plt.show();

plt.figure(figsize=(18,6));
plt.subplot(1,3,1)
plt.imshow(image[:,:,0],cmap="Greys")
plt.title('1-rst channel (hue)')
plt.subplot(1,3,2)
plt.imshow(image[:,:,1],cmap="Greys")
plt.title('2-nd channel (saturation)')
plt.subplot(1,3,3)
plt.imshow(image[:,:,2],cmap="Greys")
plt.title('3-rd channel (value)')
plt.show();
```

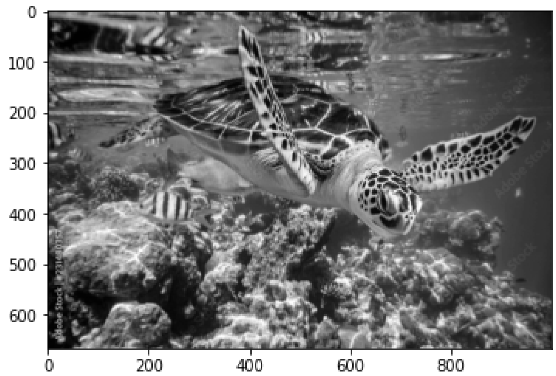




Фактически, мы можем открывать изображения прямо в нужном формате, а не проводить дополнительную предварительную обработку. Было бы полезно открыть изображение в полутоновом виде.

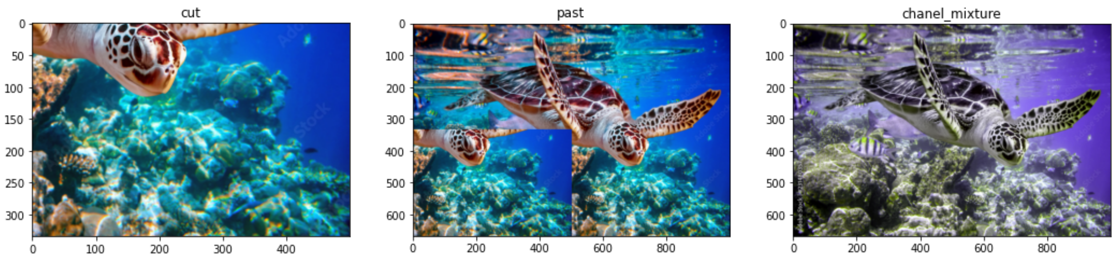
```
image = cv.imread("image.png",cv.IMREAD_GRAYSCALE)
print(image.shape)
plt.imshow(image, 'gray');
```

```
(667, 1000)
```



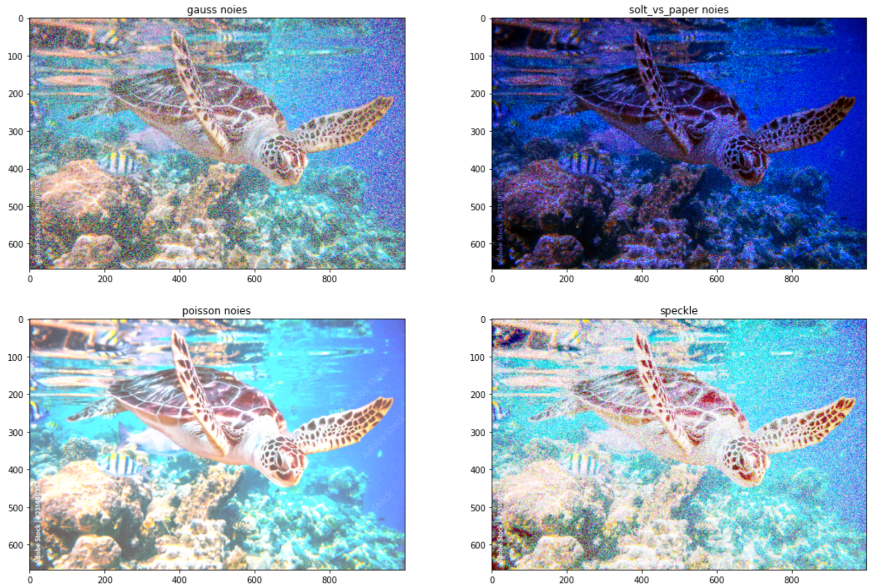
Используя opencv, вы можете работать с изображением как с матрицей (или массивом). Например, ниже приведен несколько примеров:

```
image = cv.imread("image.png",cv.IMREAD_COLOR)
image = cv.cvtColor(image, cv.COLOR_BGR2RGB)

plt.figure(figsize=(18,18))

w = image.shape[0]
h = image.shape[1]

cut = image[w//2:,h//2:,:]
plt.subplot(1,3,1); plt.imshow(cut); plt.title('cut')

past = np.copy(image)
past[w//2:,:h//2,:] = cut
plt.subplot(1,3,2); plt.imshow(past); plt.title('past')

chanel_mixture = np.copy(image)
chanel_mixture[:,:,0] = 0.1*image[:,:,0] + 0.6*image[:,:,1] +
0.3*image[:,:,2]
plt.subplot(1,3,3); plt.imshow(chanel_mixture);
plt.title('chanel_mixture')

plt.show();
```



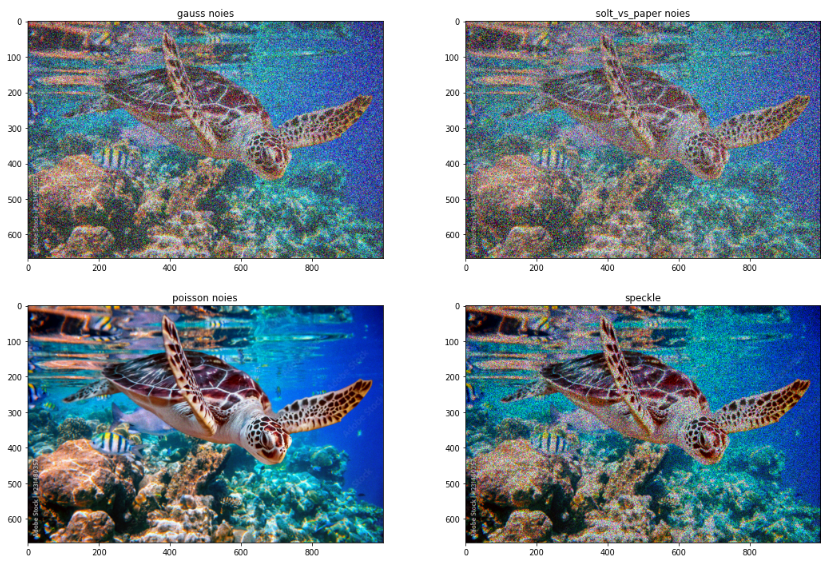
Полезно узнать, как выглядят разные типы шума. Во-первых, мы можем сделать шумы сами, используя numpy

```
image = cv.imread("image.png",cv.IMREAD_COLOR)
image = cv.cvtColor(image, cv.COLOR_BGR2RGB)

var = 10
mean = 0

#gauss
gauss_noise =
np.random.normal(mean,np.sqrt(var),image.shape).astype('uint8')
gauss = cv.add(image,gauss_noise)

#speckle
gauss_noise =
np.random.normal(mean,np.sqrt(var),image.shape).astype('uint8')
speckle = cv.add(image , cv.multiply(image , gauss_noise))

#poisson
poisson = image + np.random.poisson(var*10,image.shape)

#solt_and_paper
solt_vs_paper = 0.5
amount = 0.5
s_and_p = np.copy(image)

coords_salt = [np.random.randint(0, i - 1, int(amount * image.size *
solt_vs_paper)) for i in image.shape]
s_and_p[coords_salt] = 1

coords_pepper = [np.random.randint(0, i - 1, int(amount* image.size *
(1. - solt_vs_paper))) for i in image.shape]
s_and_p[coords_pepper] = 0


plt.figure(figsize=(18,12))
plt.subplot(221),plt.imshow(gauss);   plt.title('gauss noies')
plt.subplot(222),plt.imshow(s_and_p); plt.title('solt_vs_paper noies')
plt.subplot(223),plt.imshow(poisson); plt.title('poisson noies')
plt.subplot(224),plt.imshow(speckle); plt.title('speckle')
plt.show();
```



Также мы можем использовать готовые методы из skimage

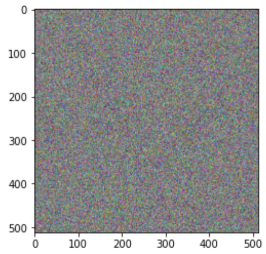
```
from skimage.util import random_noise

urllib.request.urlretrieve(image_url,"image1.png")
image = cv.imread("image.png",cv.IMREAD_COLOR)
image = cv.cvtColor(image, cv.COLOR_BGR2RGB)

s_and_p = random_noise(image, mode='s&p',salt_vs_pepper =0.5, amount =
0.5)
gauss   = random_noise(image, mode='gaussian',var = 0.3)
poisson = random_noise(image, mode='poisson',clip =0.5)
speckle = random_noise(image, mode='speckle',var = 0.7)

plt.figure(figsize=(18,12))
plt.subplot(221),plt.imshow(gauss);   plt.title('gauss noies')
plt.subplot(222),plt.imshow(s_and_p); plt.title('solt_vs_paper noies')
plt.subplot(223),plt.imshow(poisson); plt.title('poisson noies')
plt.subplot(224),plt.imshow(speckle); plt.title('speckle')
plt.show();
```



Фактически вы можете самостоятельно сгенерировать изображение в виде матрицы

```
image = np.random.randint(0,255,(512,512,3))
plt.imshow(image)
```

```
<matplotlib.image.AxesImage at 0x259778c1f48>
```



Упражнение 2

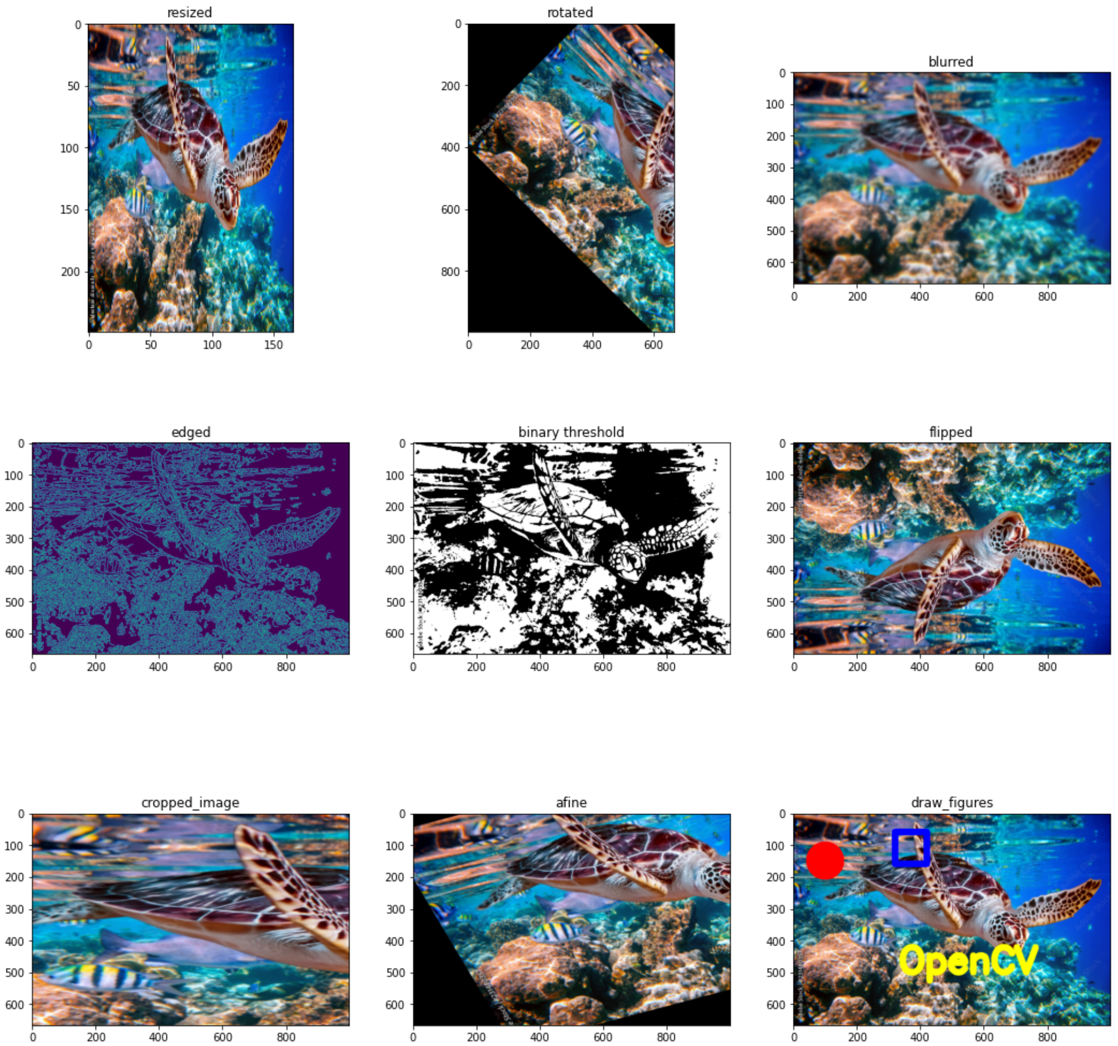
1. Нарисуйте какую нибудь фигуру и добавьте текст на созданном изображении с использованием Pillow или opencv.

OpenCV

OpenCV содержит практически все классические операции, применяемые в компьютерном зрении, ниже вы можете найти несколько примеров простых операций.

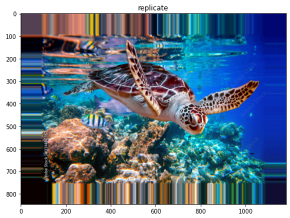
```
image = cv.imread("image.png",cv.IMREAD_COLOR)
image = cv.cvtColor(image, cv.COLOR_BGR2RGB)

plt.figure(figsize=(18,16))

w = image.shape[0]
h = image.shape[1]
center = (w // 2, h // 2)


resized = cv.resize(image, (w//4, h//4))
plt.subplot(3,3,1); plt.imshow(resized); plt.title('resized')


M = cv.getRotationMatrix2D(center, -45, 1.0)
rotated = cv.warpAffine(image, M, (w, h))
plt.subplot(3,3,2); plt.imshow(rotated); plt.title('rotated')

blurred = cv.GaussianBlur(image, (21, 21), 0)
plt.subplot(3,3,3); plt.imshow(blurred); plt.title('blurred')

edged = cv.Canny(image, 30, 150)
plt.subplot(3,3,4); plt.imshow(edged); plt.title('edged')


ret_val,threshold = cv.threshold(cv.cvtColor(image,
cv.COLOR_BGR2GRAY), 100, 255, cv.THRESH_BINARY)
```

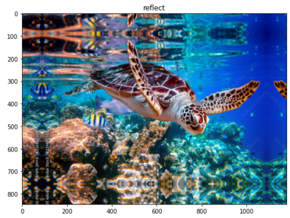
```
plt.subplot(3,3,5); plt.imshow(threshold,cmap='Greys');
plt.title('binary threshold')

flipped = cv.flip(image, flipCode=0)
plt.subplot(3,3,6); plt.imshow(flipped); plt.title('flipped')

cropped_image = image[:h // 2, w // 4:3*(w // 4),:]
cropped_image = cv.resize(cropped_image,(h,w))
plt.subplot(3,3,7); plt.imshow(cropped_image);
plt.title('cropped_image')

pts1 = np.float32([[50,50],[200,50],[50,200]])
pts2 = np.float32([[10,100],[200,50],[100,250]])
M = cv.getAffineTransform(pts1,pts2)
afine = cv.warpAffine(image,M,(h,w))
plt.subplot(3,3,8); plt.imshow(afine); plt.title('afine')

draw_figures = image.copy()
cv.rectangle(draw_figures, (320, 60), (420, 160), (0, 0, 255), 20)
cv.circle(draw_figures, (100, 150), 60, (255, 0, 0), -1)
cv.putText(draw_figures, "OpenCV ", center, cv.FONT_HERSHEY_SIMPLEX,
3.7, (255, 255, 0), 20)
plt.subplot(3,3,9); plt.imshow(draw_figures);
plt.title('draw_figures')

plt.show()
```

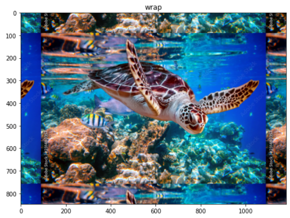


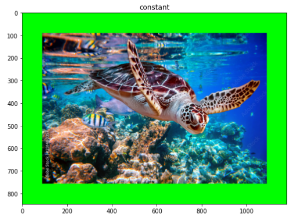
Однако, помимо методов, изученных в Pillow, OpenCV имеет много более конкретных методов. Например, ниже вы можете увидеть, как создавать границы.

```
image = cv.imread("image.png",cv.IMREAD_COLOR)
image = cv.cvtColor(image, cv.COLOR_BGR2RGB)

bs = 90 #border_size

replicate  = cv.copyMakeBorder(image,bs,bs,bs,bs,cv.BORDER_REPLICATE)
reflect    = cv.copyMakeBorder(image,bs,bs,bs,bs,cv.BORDER_REFLECT)
wrap       = cv.copyMakeBorder(image,bs,bs,bs,bs,cv.BORDER_WRAP)
constant   = cv.copyMakeBorder(image,bs,bs,bs,bs,cv.BORDER_CONSTANT,value=(0,255,0))

plt.figure(figsize=(18,12))
plt.subplot(221),plt.imshow(replicate),plt.title('replicate')
plt.subplot(222),plt.imshow(reflect),plt.title('reflect')
plt.subplot(223),plt.imshow(wrap),plt.title('wrap')
plt.subplot(224),plt.imshow(constant),plt.title('constant')
plt.show();
```



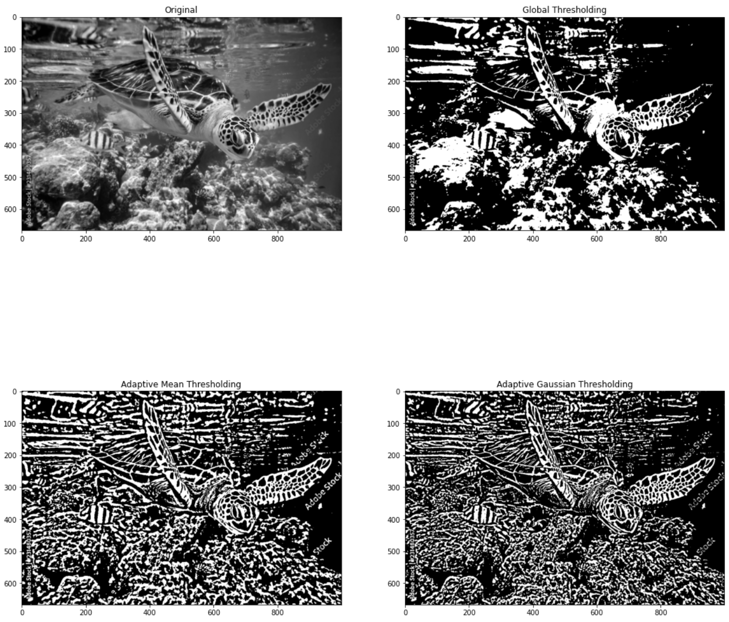






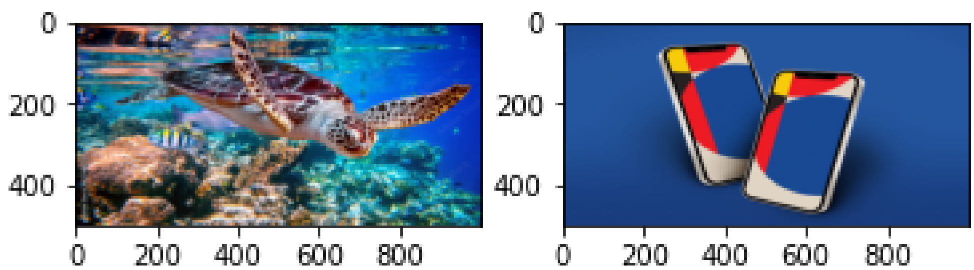
В примерах выше мы использовали глобальное пороговое значение. Но не во всех условиях это может быть правильно. Например, когда изображение имеет разное освещение в разных областях мы используем адаптивную пороговую обработку. При этом мы получаем разные пороги для разных областей одного и того же изображения.

```
image = cv.imread("image.png",cv.IMREAD_COLOR)
image = cv.cvtColor(image, cv.COLOR_BGR2GRAY)

ret,th1 = cv.threshold(image,127,255,cv.THRESH_BINARY)

th2 = cv.adaptiveThreshold(image,255,cv.ADAPTIVE_THRESH_MEAN_C,cv.THRESH_BINARY,21,-5)

th3 = cv.adaptiveThreshold(image,255,cv.ADAPTIVE_THRESH_GAUSSIAN_C,cv.THRESH_BINARY,21,-5)


plt.figure(figsize=(18,12))
plt.subplot(221),plt.imshow(image,'gray'),plt.title('Original')
plt.subplot(222),plt.imshow(th1,'gray'),plt.title('Global Thresholding')
plt.subplot(223),plt.imshow(th2,'gray'),plt.title('Adaptive Mean Thresholding')
plt.subplot(224),plt.imshow(th3,'gray'),plt.title('Adaptive Gaussian Thresholding')
plt.show();
```

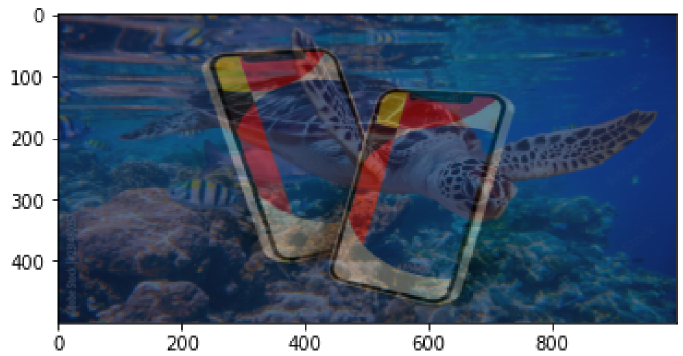


Накладывание двух изображений друг на другка

```
image1_url = 'https://as2.ftcdn.net/v2/jpg/02/31/48/03/1000_F_231480357_TGpMz4r5HSFAlm43FkZ366FjFZuuoRA8.jpg'
urllib.request.urlretrieve(image1_url,"image1.png")
image1 = cv.imread("image1.png",cv.IMREAD_COLOR)
image1 = cv.cvtColor(image1, cv.COLOR_BGR2RGB)
image1 = cv.resize(image1, (1000,500))

image2_url = 'https://as2.ftcdn.net/jpg/04/46/10/77/1024W_F_446107734_NaiNx4Qxq1bT4ipU1vGohw1dtU9kOWpx_NW1.jpg'
urllib.request.urlretrieve(image2_url,"image2.png")
image2 = cv.imread("image2.png",cv.IMREAD_COLOR)
image2 = cv.cvtColor(image2, cv.COLOR_BGR2RGB)
image2 = cv.resize(image2, (1000,500))

plt.subplot(1,2,1)
plt.imshow(image1)
plt.subplot(1,2,2)
plt.imshow(image2)
plt.show();

mix = cv.addWeighted(image1,0.3,image2,0.4,0)
plt.imshow(mix)
```



`<matplotlib.image.AxesImage at 0x2590f8131c8>`

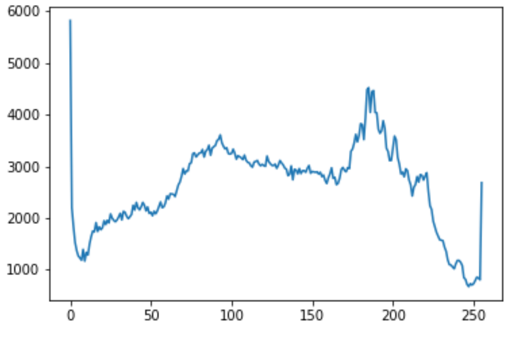


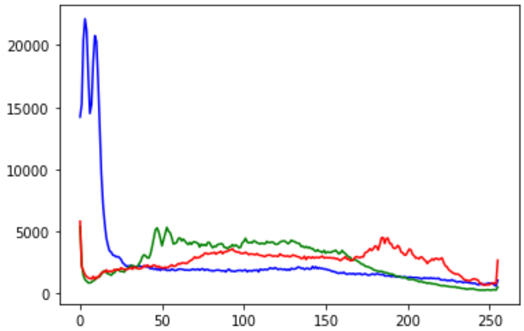
Упражнение 3

1. Выберите любое изображение и попробуйте проделать с ним не менее 5 операций с помощью opencv.

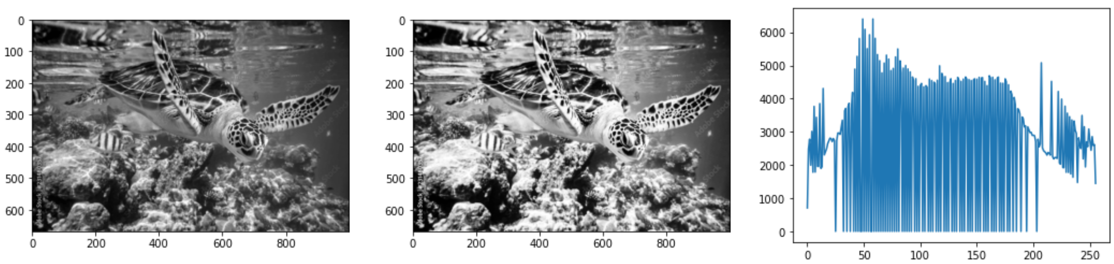
Работа с гистограммами

```
image = cv.imread("image.png",cv.IMREAD_COLOR)
image = cv.cvtColor(image, cv.COLOR_BGR2RGB)


hist = cv.calcHist([image],[2],None,[256],[0,256])
plt.plot(hist); plt.show()

color = ('b','g','r')

for chanel_i, color_i in enumerate(color):
    hist = cv.calcHist([image],[chanel_i],None,[256],[0,256])
    plt.plot(hist, c= color_i)
```






```
image = cv.imread("image.png",cv.IMREAD_COLOR)
image = cv.cvtColor(image, cv.COLOR_BGR2GRAY)

image_eq = cv.equalizeHist(image)
hist = cv.calcHist([image_eq],[0],None,[256],[0,256])

plt.figure(figsize=(18,4))
plt.subplot(1,3,1); plt.imshow(image, 'gray');
plt.subplot(1,3,2); plt.imshow(image_eq, 'gray');
plt.subplot(1,3,3); plt.plot(hist);
plt.show();
```



Упражение 4

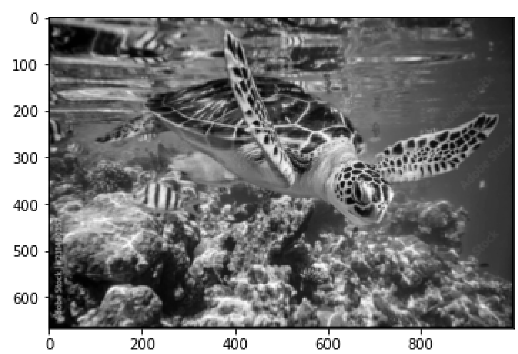
1. Изучите гистограмму бинарного изображения,
2. Изучите гистограмму изображения, полученную с помощью пороговых функций THRESH_TOZERO и THRESH_TRUNC.
3. Выберите как минимум 3 операции из предыдущего раздела (простые операции в opencv) и изучите гистограмму для всех из них. Не забывайте преобразовывать изображения в оттенки серого.
4. Изучите, как изменение гистограммы зависит от величины шума.

Операция свертки

Свертки - это математические операции между двумя функциями, которые создают третью функцию. При обработке изображений эта операция осуществляется путем комбинации пространственного сдвига и операции умножения одного изображеия на другое. Одно из бражений называется ядро. Ядра определяет размер свертки, применяемые веса и точку привязки, обычно расположенную в центре ядра. Свертка - основная операция в большинстве методов компьютерного зрения. Для начала попробуем сделать это в формате numpy.

- Примечание * Для получения изображения того же размера нам нужно будет сделать расширение изображения нулями.

```
def convolve2D(image, kernel):

    kernel = np.asarray(kernel)

    # for obtain
    image_pad = np.pad(image,pad_width=(kernel.shape[0]//2-1,
kernel.shape[1]//2-1 ),mode='constant')

    output    = np.zeros_like(image_pad)

    for x in range(kernel.shape[0]//2,
image_pad.shape[0]-kernel.shape[0]):
        for y in range(kernel.shape[1]//2,
image_pad.shape[1]-kernel.shape[1]):

            image_path = image[x: x + kernel.shape[1],
                               y: y + kernel.shape[0]]

            output[x, y] = np.sum(kernel*image_path)


    return output

image = cv.imread("image.png",cv.IMREAD_COLOR)
image = cv.cvtColor(image, cv.COLOR_BGR2GRAY)

kernel = np.asarray([[1,1,1],[1,1,1],[1,1,1]])/9

out = convolve2D(image,kernel)

plt.imshow(out, 'gray');
plt.show();
print(out.shape, image.shape)
```

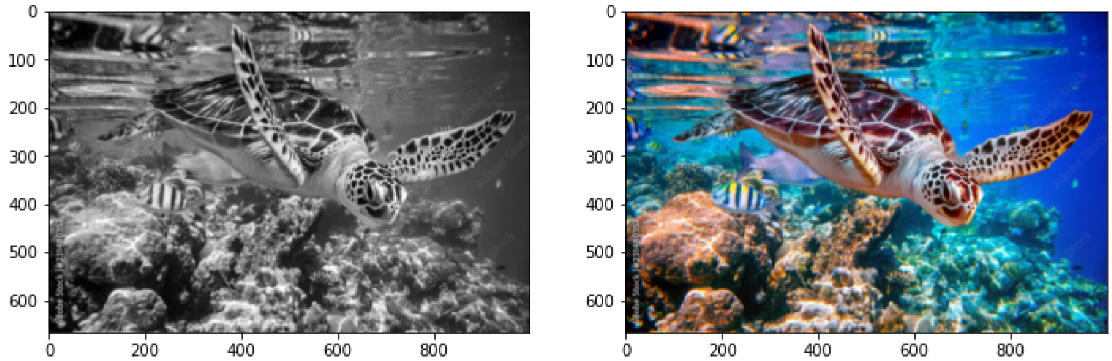



```
(667, 1000) (667, 1000)
```

Теперь давайте протестируем ту же операцию с opencv

```
image = cv.imread("image.png",cv.IMREAD_COLOR)
image_rgb  = cv.cvtColor(image, cv.COLOR_BGR2RGB)
```

```python
image_gray = cv.cvtColor(image, cv.COLOR_BGR2GRAY)

kernel = np.asarray([[1,1,1],[1,1,1],[1,1,1]])/9

out_gray = cv.filter2D(src=image_gray, ddepth=-1, kernel=kernel)
out_rgb  = cv.filter2D(src=image_rgb,  ddepth=-1, kernel=kernel)
plt.figure(figsize=(12,6))
plt.subplot(1,2,1); plt.imshow(out_gray, 'gray');
plt.subplot(1,2,2); plt.imshow(out_rgb);
plt.show();
print(out_gray.shape, image.shape)
```

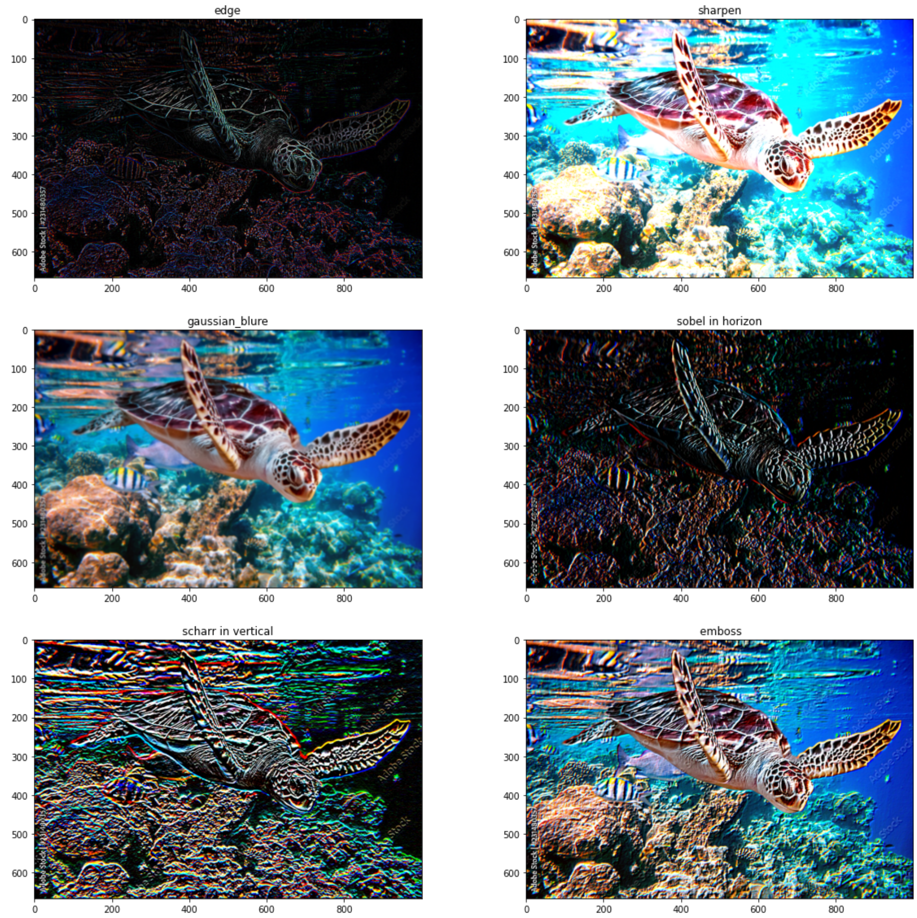



```
(667, 1000) (667, 1000, 3)
```

Давайте изучим несколько простых фильтров

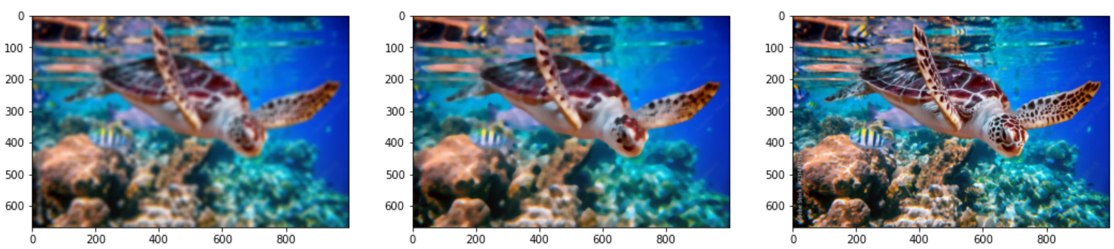
```python
image = cv.imread("image.png",cv.IMREAD_COLOR)
image = cv.cvtColor(image, cv.COLOR_BGR2RGB)

#Edge detection
kernel = np.array([[-1, -1,  -1],
                   [-1,  8,  -1],
                   [-1, -1,  -1]])
edge  = cv.filter2D(src=image,  ddepth=-1, kernel=kernel)


#Sharpen
kernel = np.array([[ 0, -1,   0],
                   [-1,  6,  -1],
                   [ 0, -1,   0]])
sharpen  = cv.filter2D(src=image,  ddepth=-1, kernel=kernel)

#Gaussian Blure
kernel =
[[2,4,5,4,2],[4,9,12,9,4],[5,12,15,12,5],[4,9,12,9,4],[2,4,5,4,2]]
kernel = np.asarray(kernel)/np.sum(kernel)
gaussian_blure  = cv.filter2D(src=image,  ddepth=-1, kernel=kernel)

#Sobel edge detection in horizon
kernel = np.array([[-1, 0,  1],
                   [-2, 0,  2],
```

```python
                   [-1, 0,  1]])
sobelx  = cv.filter2D(src=image,  ddepth=-1, kernel=kernel)

#Scharr edge detection in vertical
kernel = np.array([[-3, -10, -3],
                   [ 0,  0,  0],
                   [ 3,  10,  3]])
scharry  = cv.filter2D(src=image,  ddepth=-1, kernel=kernel)

#  emboss filter
kernel = np.array([[-2, -1, 0],
                   [-1,  1, 1],
                   [ 0,  1, 2]])
kernel = np.asarray(kernel)/np.sum(kernel)
emboss  = cv.filter2D(src=image,  ddepth=-1, kernel=kernel)


plt.figure(figsize=(18,18))
plt.subplot(3,2,1); plt.imshow(edge, 'gray'); plt.title("edge")
plt.subplot(3,2,2); plt.imshow(sharpen, 'gray'); plt.title("sharpen")
plt.subplot(3,2,3); plt.imshow(gaussian_blure, 'gray');
plt.title("gaussian_blure")
plt.subplot(3,2,4); plt.imshow(sobelx, 'gray'); plt.title("sobel in
horizon")
plt.subplot(3,2,5); plt.imshow(scharry, 'gray'); plt.title("scharr in
vertical")
plt.subplot(3,2,6); plt.imshow(emboss, 'gray');plt.title(" emboss")
plt.show();
```



На самом деле OpenCV содержит некоторые фильтры как встроенные функции.

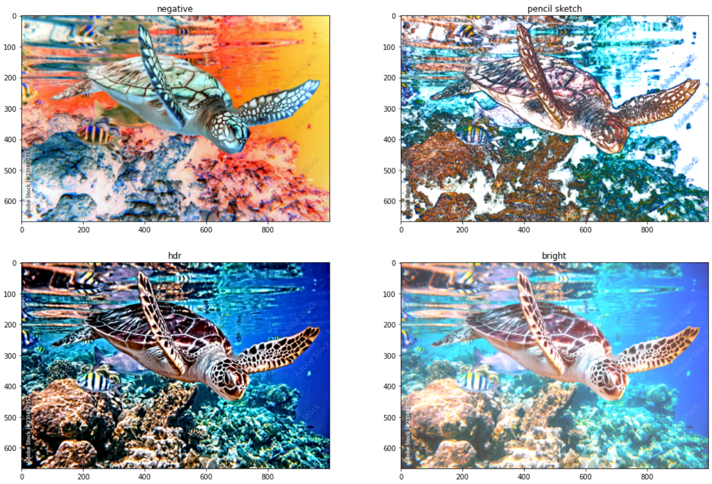
```
gauss_blure  = cv.GaussianBlur(image_rgb, (15,15), 10.4)
median_blure = cv.medianBlur(image_rgb, 15)
bilateral    = cv.bilateralFilter(image_rgb,9,75,75)

plt.figure(figsize=(18,6))
plt.subplot(1,3,1); plt.imshow(gauss_blure);
plt.subplot(1,3,2); plt.imshow(median_blure);
plt.subplot(1,3,3); plt.imshow(bilateral);
plt.show();
```



Также отметим, что opencv содержит множество более сложных операций, подобных фильтру.

```
sk_gray, sk_color = cv.pencilSketch(image, sigma_s=6, sigma_r=0.07,
shade_factor=0.1)
hdr = cv.detailEnhance(image, sigma_s=12, sigma_r=0.15)
inverse = cv.bitwise_not(image)
bright  = cv.convertScaleAbs(image, beta=70)
```

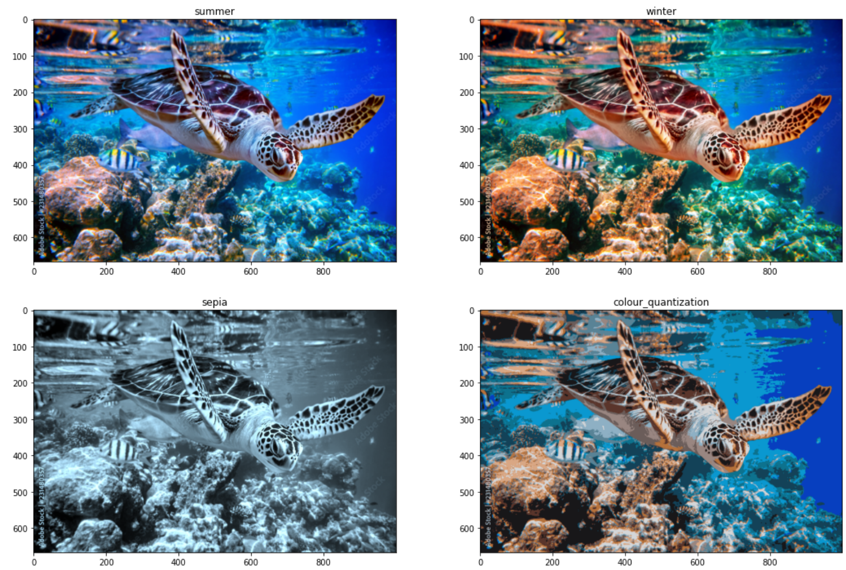
```
plt.figure(figsize=(18,12))
plt.subplot(2,2,1); plt.imshow(inverse);  plt.title("negative")
plt.subplot(2,2,2); plt.imshow(sk_color); plt.title("pencil sketch")
plt.subplot(2,2,3); plt.imshow(hdr);      plt.title("hdr")
plt.subplot(2,2,4); plt.imshow(bright);   plt.title("bright")

plt.show();
```




```
from scipy.interpolate import UnivariateSpline

image = cv.imread("image.png",cv.IMREAD_COLOR)
image = cv.cvtColor(image, cv.COLOR_BGR2RGB)

def LookupTable(x, y):
    spline = UnivariateSpline(x, y)
    return spline(range(256))

def summer(img):

    increaseLookupTable = LookupTable([0, 64, 128, 256], [0, 80, 160,
256])
    decreaseLookupTable = LookupTable([0, 64, 128, 256], [0, 50, 100,
256])

    blue_channel, green_channel,red_channel  = cv.split(img)

    red_channel  = cv.LUT(red_channel,
increaseLookupTable).astype(np.uint8)
    blue_channel = cv.LUT(blue_channel,
decreaseLookupTable).astype(np.uint8)

    out= cv.merge((blue_channel, green_channel, red_channel ))

    return out

def winter(img):
    increaseLookupTable = LookupTable([0, 64, 128, 256], [0, 80, 160,
256])
```

```
    decreaseLookupTable = LookupTable([0, 64, 128, 256], [0, 50, 100,
256])

    blue_channel, green_channel,red_channel = cv.split(img)

    red_channel  = cv.LUT(red_channel,
decreaseLookupTable).astype(np.uint8)
    blue_channel = cv.LUT(blue_channel,
increaseLookupTable).astype(np.uint8)

    out= cv.merge((blue_channel, green_channel, red_channel))

    return out

def sepia(img):
    img_sepia = np.array(img, dtype=np.float64) # converting to float
to prevent loss
    img_sepia = cv.transform(img_sepia,
                             np.matrix([[0.272, 0.534, 0.131],
                                        [0.349, 0.686, 0.168],
                                        [0.393, 0.769, 0.189]])) #
multipying image with special sepia matrix
    img_sepia[np.where(img_sepia > 255)] = 255 # normalizing values
greater than 255 to 255
    img_sepia = np.array(img_sepia, dtype=np.uint8)
    return img_sepia

def edge_mask(img, line_size=3, blur_value=3):
    gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
    gray_blur = cv.medianBlur(gray, blur_value)
    edges = cv.adaptiveThreshold(gray_blur, 255,
cv.ADAPTIVE_THRESH_MEAN_C, cv.THRESH_BINARY, line_size, blur_value)
    return edges


def countours(image):
    contoured_image = image
    gray = cv.cvtColor(contoured_image, cv.COLOR_BGR2GRAY)
    edged = cv.Canny(gray, 120, 200)
    contours, hierarchy = cv.findContours(edged, cv.RETR_EXTERNAL,
cv.CHAIN_APPROX_NONE)[-2:]
    cv.drawContours(contoured_image, contours, contourIdx=-1, color=6,
thickness=1)
    return contoured_image

def colour_quantization(image, K=13):
    Z = image.reshape((-1, 3))
    Z = np.float32(Z)
    criteria = (cv.TERM_CRITERIA_EPS + cv.TERM_CRITERIA_MAX_ITER, 100,
0.001)
    compactness, label, center = cv.kmeans(Z, K, None, criteria, 1,
```

```
cv.KMEANS_RANDOM_CENTERS)
    center = np.uint8(center)
    res = center[label.flatten()]
    res2 = res.reshape((image.shape))
    return res2

plt.figure(figsize=(18,12))
plt.subplot(2,2,1); plt.imshow(summer(image)); plt.title("summer")
plt.subplot(2,2,2); plt.imshow(winter(image)); plt.title("winter")
plt.subplot(2,2,3); plt.imshow(sepia(image));  plt.title("sepia")
plt.subplot(2,2,4); plt.imshow(colour_quantization(image));
plt.title("colour_quantization")
plt.show()
```



Упражнеие 5

1. Изучите, как работает фильтр с ядром `kernel = np.ones ((25,25)); kernel = kernel / np.max (kernel)`.
2. Попробуйте несколько функций в opencv, например `cv.stylization (image, sigma_s = 20, sigma_r = 0.6)` или `cv.detailEnhance (image, sigma_s = 10, sigma_r = 0.15)` для изучаемого изображения.
3. Выберите изображение и добавьте шумы (выберите один тип), затем попробуйте улучшить качество обработкой показанной выше.
4. Изучите изменения гистограммы для изображений с разной фильтрацией (минимум 5 типов).
5. Создайте эффект мультипликции с помощью функций `countours()` и `colour_quantization()`, приведенных выше.

Лабораторная 2

Работа сопровождается документом в формате ipnb

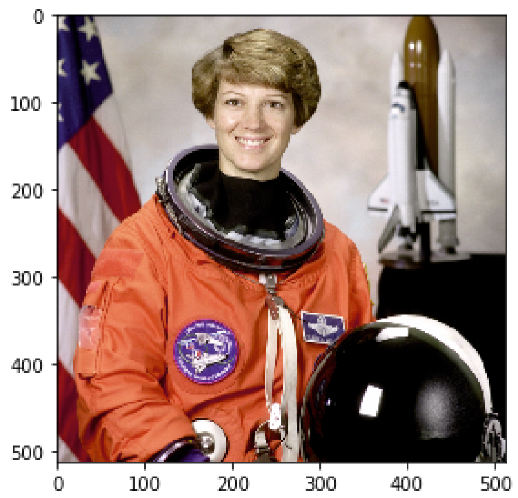
Изучение особенностей классических методов решения задач компьютерного зрения.

Изучение особенностей классических методов решения задач компьютерного зрения. Методы HOG, DAISY, watershed, детекция углов, корреляция и других.

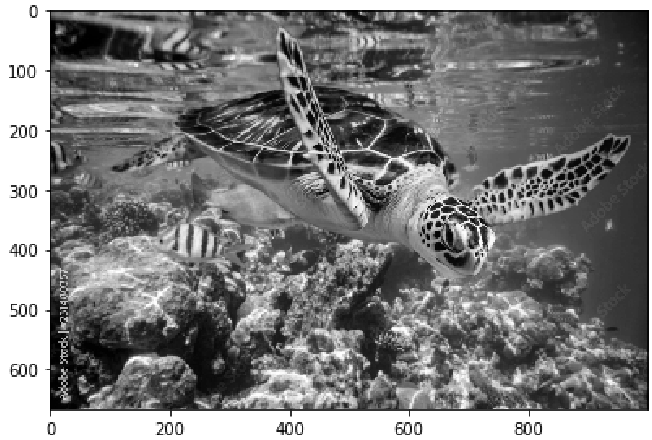
пакет библиотек scikit-image

Scikit-image или skimage - это мощный пакет (или инструмент) Python с открытым исходным кодом, предназначенный для решения задач предварительной обработки изображений и компьютерного зрения с использованием классических подходов. Для начала изучим основы работы с изображениями.

```
import skimage

from skimage.io import imread, imshow

import skimage.data

from skimage import color
from skimage import transform
from skimage import exposure
from skimage import filters
from skimage import morphology
from skimage import feature
from skimage import segmentation

import urllib.request
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

image = skimage.data.astronaut()

imshow(image);
```



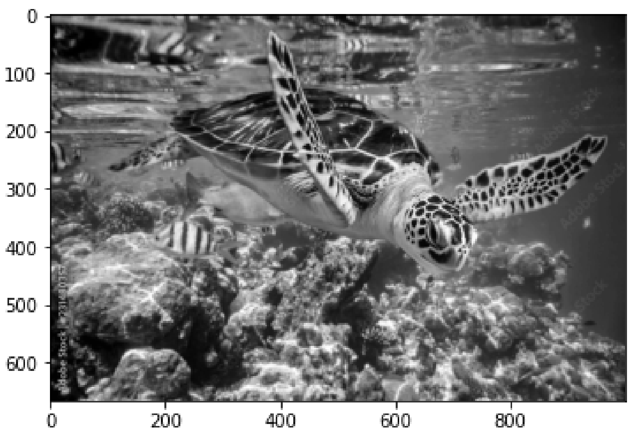
```
image_url = 'https://as2.ftcdn.net/v2/jpg/02/31/48/03/1000_F_231480357_TGpMz4r5HSFAlm43FkZ366FjFZuuoRA8.jpg'

urllib.request.urlretrieve(image_url,"image.png")

image_gray = imread('image.png', as_gray=True)
imshow(image_gray);
```



На самом деле изображения на выходе imread имеют тип массива numpy. Таким образом, мы можем работать с ними в любых других пакетах без дополнительных преобразований.

```
print(type(image_gray))
plt.imshow(image_gray,'gray');
```

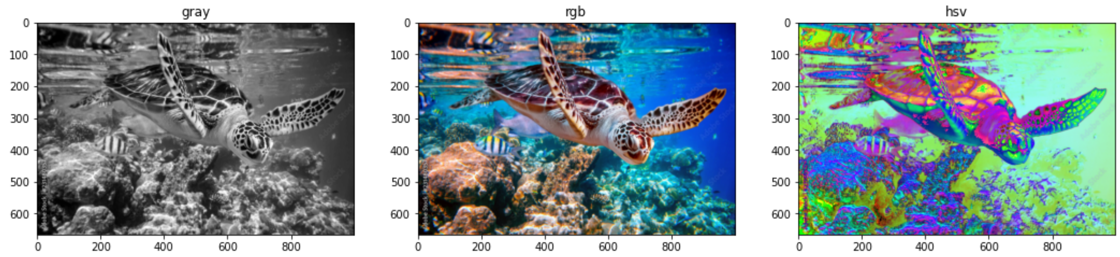
```
<class 'numpy.ndarray'>
```



Как и в предыдущей работе, здесь мы можем преобразовывать изображения из одного формата в другой с помощью встроенных функций.

```
image_rgb  = imread('image.png', as_gray=False)
image_gray = color.rgb2gray(image_rgb)
img_hsv    = color.rgb2hsv(image_rgb)

plt.figure(figsize=(18,12))
plt.subplot(131),plt.imshow(image_gray); plt.title('gray')
plt.subplot(132),plt.imshow(image_rgb);  plt.title('rgb')
```

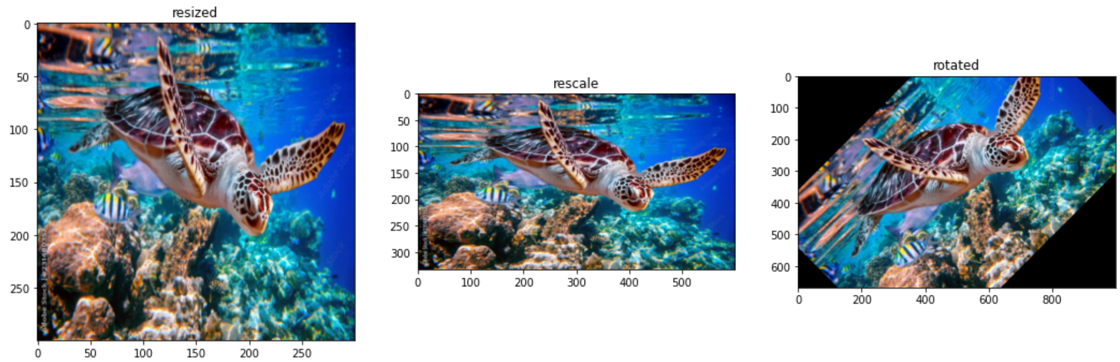
```
plt.subplot(133),plt.imshow(img_hsv);    plt.title('hsv')
plt.show();
```



skimage содержит множество популярных методов преобразования изображений.

```
image = imread('image.png', as_gray=False)

resized = transform.resize(image, (300, 300))
rescale = transform.rescale(image, scale=(0.5, 0.6))
rotated = transform.rotate(image, angle=45)

plt.figure(figsize=(18,12))
plt.subplot(131),plt.imshow(resized); plt.title('resized')
plt.subplot(132),plt.imshow(rescale); plt.title('rescale')
plt.subplot(133),plt.imshow(rotated); plt.title('rotated')
plt.show();
```

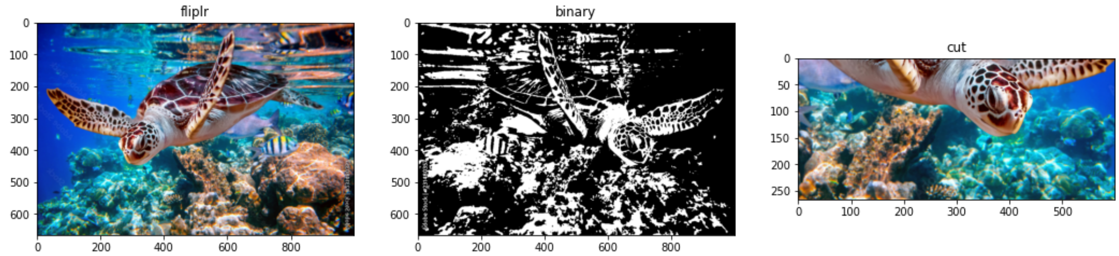
```
C:\ProgramData\Anaconda3\lib\site-packages\skimage\transform\_warps.py
:23: UserWarning: The default multichannel argument (None) is
deprecated.  Please specify either True or False explicitly.
multichannel will default to False starting with release 0.16.
  warn('The default multichannel argument (None) is deprecated.
Please '
```



Однако для некоторых из них нужно использовать другие пакеты.

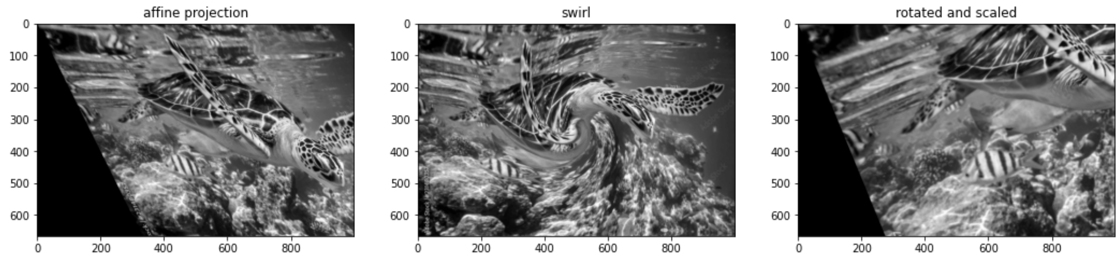
```
image  = imread('image.png', as_gray=False)

#binary image
def rgb2gray(image):
    rgb2gray_weights = [0.2989, 0.5870, 0.1140]
    image_gray = np.dot(image[...,:3], rgb2gray_weights) # sum of
weighted channels
    return image_gray

def rgb2binary(image, threshold = 127):
```

```
    binary = rgb2gray(image)
    binary = binary >= threshold # if >=127, then 1, else 0
    return binary

fliplr = np.fliplr(image)
binary = rgb2binary(image)
cut    = image[300:(image.shape[0]-100), 300:(image.shape[1]-100), :]

plt.figure(figsize=(18,12))
plt.subplot(131),plt.imshow(fliplr);          plt.title('fliplr')
plt.subplot(132),plt.imshow(binary, 'gray'); plt.title('binary')
plt.subplot(133),plt.imshow(cut);             plt.title('cut')
plt.show();
```



Вот еще несколько примеров преобразований

```
image  = imread('image.png', as_gray=True)


swirl  = transform.swirl(image, rotation=0, strength=3, radius=800)

tform   = transform.SimilarityTransform(scale=0.6, rotation=np.pi/8)
rotated = transform.warp(image, tform)

tform  = transform.AffineTransform(shear=np.pi/6)
affine = transform.warp(image, tform)

plt.figure(figsize=(18,18))
plt.subplot(131),plt.imshow(affine, 'gray');   plt.title('affine
projection')
plt.subplot(132),plt.imshow(swirl, 'gray'); plt.title('swirl')
plt.subplot(133),plt.imshow(rotated, 'gray'); plt.title('rotated and
scaled')
plt.show()
```



Также Skimage содержит множество популярных фильтров и других методов обработки изображений.

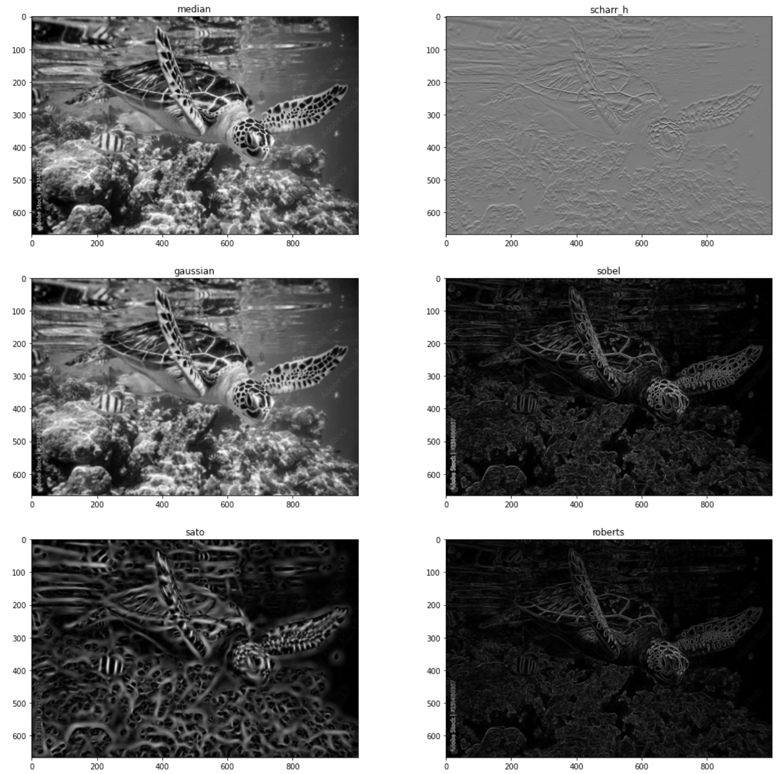
```
image  = imread('image.png', as_gray=False)

bright    = exposure.adjust_gamma(image, gamma=0.3,gain=1)
dark      = exposure.adjust_gamma(image, gamma=1.8,gain=1)
equalized = exposure.equalize_hist(image, nbins=3)

plt.figure(figsize=(18,12))
plt.subplot(131),plt.imshow(bright);    plt.title('bright')
plt.subplot(132),plt.imshow(dark);      plt.title('dark')
plt.subplot(133),plt.imshow(equalized); plt.title('equalized')
plt.show();
```

```
C:\ProgramData\Anaconda3\lib\site-packages\skimage\exposure\exposure.p
y:124: UserWarning: This might be a color image. The histogram will be
computed on the flattened image. You can instead apply this function
to each color channel.
  warn("This might be a color image. The histogram will be "
```

Давайте протестируем фильтры, однако мы должны отметить, что в «skimage» по умолчанию для большинства фильтров требуются представить изображения в градациях серого.

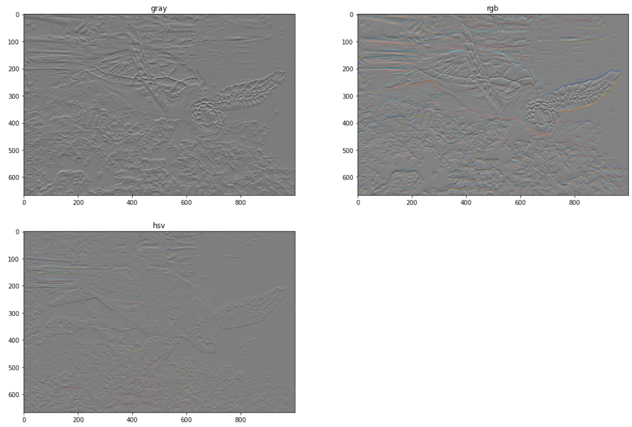
```
image  = imread('image.png', as_gray=True)

median   = filters.median(image)
scharr_h = filters.scharr_h(image)
gaussian = filters.gaussian(image)
sobel    = filters.sobel(image)
sato     = filters.sato(image)
roberts  = filters.roberts(image)

plt.figure(figsize=(18,18))
plt.subplot(321),plt.imshow(median, 'gray');   plt.title('median')
plt.subplot(322),plt.imshow(scharr_h, 'gray'); plt.title('scharr_h')
plt.subplot(323),plt.imshow(gaussian, 'gray'); plt.title('gaussian')
plt.subplot(324),plt.imshow(sobel,   'gray');  plt.title('sobel')
plt.subplot(325),plt.imshow(sato, 'gray');     plt.title('sato')
plt.subplot(326),plt.imshow(roberts, 'gray');  plt.title('roberts')
plt.show();
```

```
C:\ProgramData\Anaconda3\lib\site-packages\skimage\util\dtype.py:135:
UserWarning: Possible precision loss when converting from float64 to
uint8
  .format(dtypeobj_in, dtypeobj_out))
```



Однако все фильтры могут быть адаптированы к rgb или другим форматам.

```
from skimage.color.adapt_rgb import adapt_rgb, each_channel, hsv_value

@adapt_rgb(each_channel)
def sobel_rgb(image):
    return filters.sobel_h(image)

@adapt_rgb(hsv_value)
def sobel_hsv(image):
    return filters.sobel_h(image)

image_rgb  = imread('image.png', as_gray=False)
image_gray = color.rgb2gray(image_rgb)
image_hsv  = color.rgb2hsv(image_rgb)


image_gray = filters.sobel_h(image_gray)
image_rgb  = exposure.rescale_intensity(1-sobel_rgb(image_rgb))
image_hsv  = exposure.rescale_intensity(1-sobel_hsv(image_hsv))

plt.figure(figsize=(18,12))
plt.subplot(221),plt.imshow(image_gray, 'gray');   plt.title('gray')
plt.subplot(222),plt.imshow(image_rgb); plt.title('rgb')
plt.subplot(223),plt.imshow(image_hsv); plt.title('hsv')
plt.show()
```



Упражение 1

1. Выберите изображение и попробуйте как минимум 5 техник, показанных выше.
2. Добавьте различные типы шумов к вашему изображению (см. Предыдущие занятие) и попробуйте сделать шумоподавление, используя некоторые техники показаные тут.

Работа с признаками изображений

Помимо методов обработки изображений, skimage содержит множество методов для решения простых задач компьютерного зрения, ниже вы можете найти несколько их примеров. Сначала нам нужно будет выбрать некоторые функции.

```
from skimage import feature

#Auxiliary function
def plt_feature(image, name = '', rescale =(0, 23) ):
    if rescale:
        image = exposure.rescale_intensity(image,
                                            in_range=rescale)
    plt.figure(figsize=(14,14));
    plt.imshow(image, 'gray');
    plt.title(name);
    plt.show()
```

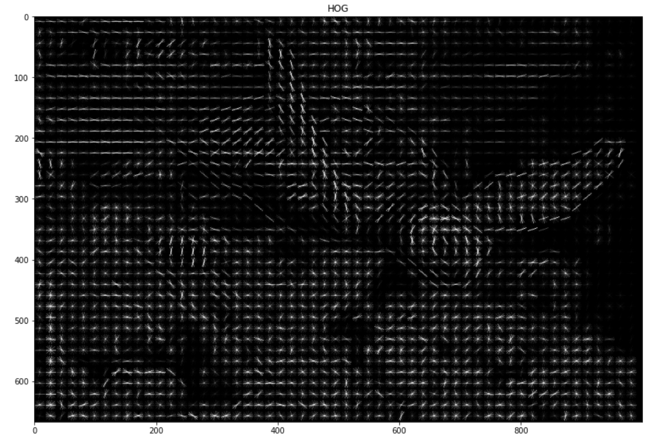
Выделение локальных признаков

«Гистограмма ориентированного градиента» (HOG) является одним из самых популярных методов работы с признаками во многих задачах компьютерного зрения, в частности для обнаружения объектов.

Обзор алгоритма HOG   (необязательно) глобальная нормализация изображения - уменьшить эффекты освещения. вычисление градиентного изображения в направлениях x и y для каждого пикселя - для захвата контура, силуэта или информации о текстуре, разделить изображение на небольшие пространственные области (ячейки, блоки) с предопределенным окном, вычисление гистограмм градиента в ячейке - кодирование локальных характеристик в виде 1d гистограмм ориентации через каждый блок (ячейку), нормализация по блокам преобразование результатов в вектор признаков

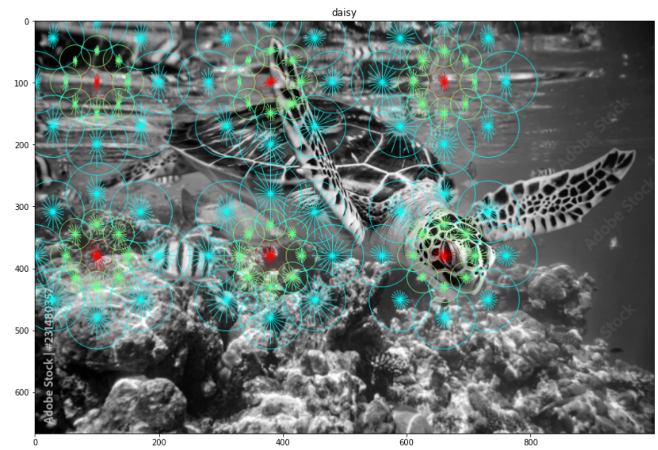
```
image   = imread('image.png', as_gray=False)
```

```python
fd, hog_image = feature.hog(image,
                            pixels_per_cell=(18, 18),
                            visualize=True)

plt_feature(hog_image,'HOG')
```



Существует множество методов, основанных на функциях HOG, один из них - DAISY. Он разработан таким образом, чтобы выделять плотность объектов на изображения, что полезно, например, для представление изображений "мешок признаков".

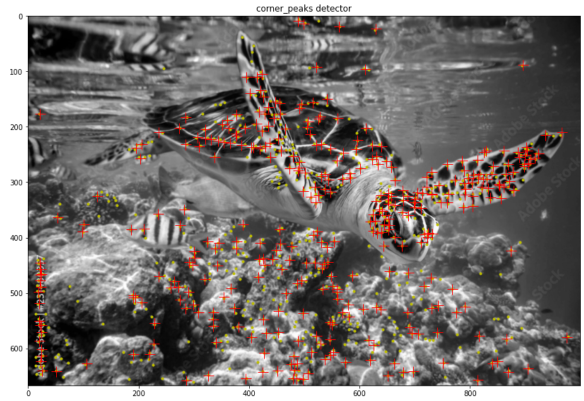
```python
image  = imread('image.png', as_gray=True)

fd, daisy = feature.daisy(image,
                          step=280,
                          radius=100,
                          rings=2,
                          orientations=20,
                          visualize=True)
plt_feature(daisy,'daisy',None)
```



Для распознавания образов изображения также может быть полезно искать углы.

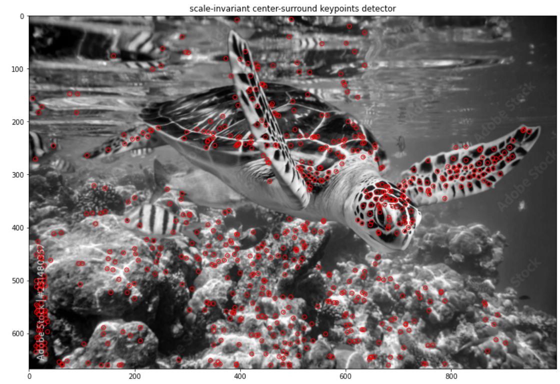
```python
# Sheared checkerboard
image  = imread('image.png', as_gray=True)

coords = feature.corner_peaks(feature.corner_harris(image),
min_distance=5, threshold_rel=0.01)
coords_subpix = feature.corner_subpix(image, coords, window_size=13)
```

```
plt.figure(figsize=(14,14))

plt.imshow(image, 'gray')

plt.plot(coords[:, 1], coords[:, 0], color='y',
marker='o',linestyle='None', markersize=3)

plt.plot(coords_subpix[:, 1], coords_subpix[:, 0], '+r',
markersize=15)

plt.title('corner_peaks detector')

plt.show()
```



Другой тип детектора признаков - это детектор признаков CENSURE (детектор ключевых точек, не зависящий от масштаба).

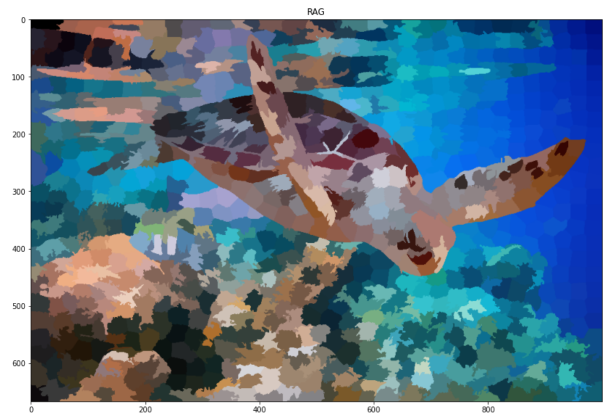
```
image  = imread('image.png', as_gray=True)

detector = feature.CENSURE()

detector.detect(image)

plt.figure(figsize=(14,14))

plt.imshow(image, cmap=plt.cm.gray)

plt.scatter(detector.keypoints[:, 1],
            detector.keypoints[:, 0],
            facecolors='none',
            edgecolors='r')

plt.title('scale-invariant center-surround keypoints detector')
plt.show()
```



Сегментация

В приведенных выше примерах показаны примеры обнаружения локальных особенностей объектов на изображениях. Однако в некоторых случаях сегментация объектов может оказаться более полезной. Сегментация пороговых значений RAG В этом примере создается график смежности регионов (RAG). Регионы объединяются в области, похожие по цвету. Мы строим RAG и определяем края как разницу в среднем цвете. Затем мы объединяем области с похожими средними цветами.

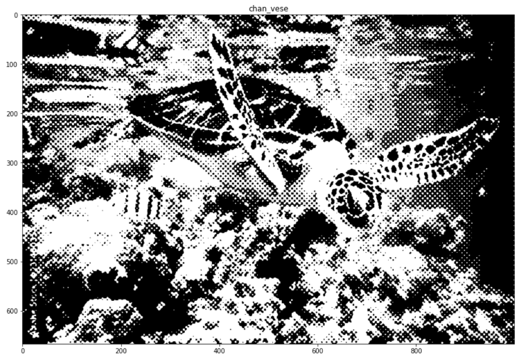
```
image =  imread('image.png', as_gray=False)

labels = segmentation.slic(image, compactness=30, n_segments=900)
RAG = color.label2rgb(labels, image, kind='avg', bg_label=0)

plt_feature(RAG,'RAG',None)
```



Алгоритм сегментации Чан-Весе предназначен для сегментации объектов без четко определенных границ. Этот алгоритм основан на наборах уровней, которые итеративно изменяются для минимизации целевой функции - энергии, которая определяется взвешенными значениями, соответствующими сумме различий интенсивности от среднего значения за пределами сегментированной области к сумме отличий от среднего значения внутри сегментированной области, и слогаемым которое зависит от длины границы сегментированной области.

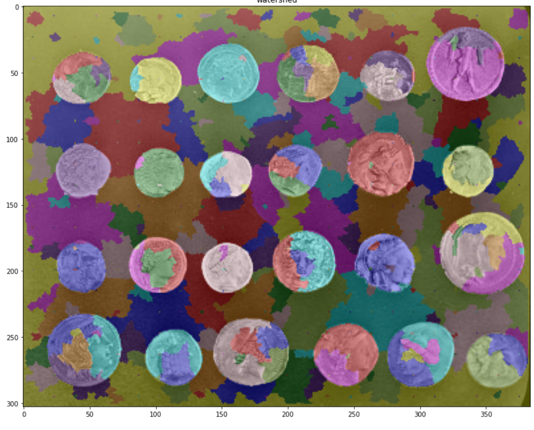
```
image =  imread('image.png', as_gray=True)
imagef = skimage.img_as_float(image)


chan_vese = segmentation.chan_vese(image, tol=0.01, max_iter=100,)
```

```python
plt_feature(chan_vese,'chan_vese',None)
```



Watershed Другой тип классических алгоритмов компьютерного зрения - это сегментация водораздела. Watershed - это классический алгоритм, используемый для сегментации, то есть для разделения различных объектов на изображении. Алгоритм рассматривает значения пикселей как местную топографию (высоту). Во многих случаях алгоритм начинается с маркеров, которые могут быть выбраны в качестве локальных минимумов значений изображения. Алгоритм затопляет бассейны от маркеров до тех пор, пока бассейны, относящиеся к разным маркерам, не встретятся на линиях водоразделов.

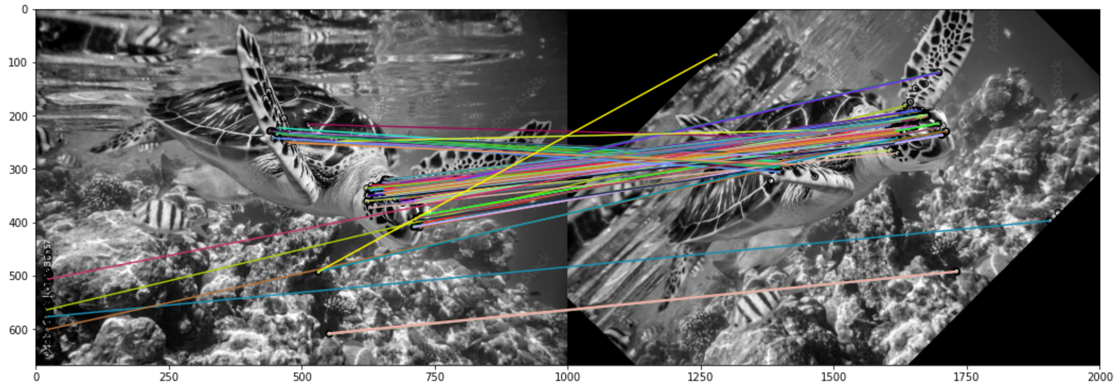
```python
coins = data.coins()

edges = filters.sobel(coins)

grid = skimage.util.regular_grid(coins.shape, n_points=468)

seeds = np.zeros(coins.shape, dtype=int)
seeds[grid] = np.arange(seeds[grid].size).reshape(seeds[grid].shape) +
1

watershed = segmentation.watershed(edges, seeds)

plt_feature(color.label2rgb(watershed, coins,
bg_label=-1),'watershed',None)
```



Корреляция и сопоставление признаков

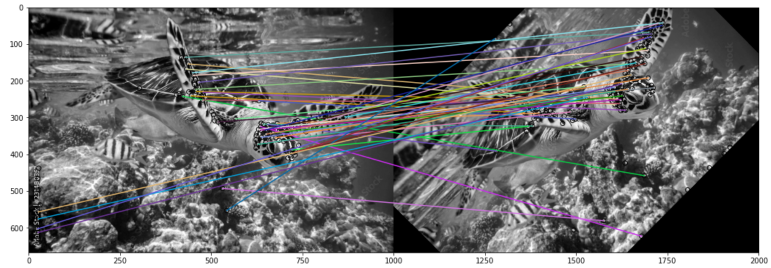
Помимо неконтролируемого извлечения признаков, в некоторых случаях вам потребуется сопоставить особенности нескольких изображений или изображений с шаблонами. Ниже приведены несколько методов сопоставления признаков.

```
image1 = imread('image.png', as_gray=True)

image2 = transform.rotate(image1, 45) # distorted image

descr = feature.ORB(n_keypoints=200) #descriptor_extractor

# original image features
descr.detect_and_extract(image1)
keypoints1 = descr.keypoints
descr1     = descr.descriptors

# distorted image features
descr.detect_and_extract(image2)
keypoints2 = descr.keypoints
descr2     = descr.descriptors

# matches features
matches12 = feature.match_descriptors(descr1, descr2,
cross_check=True)

fig, ax = plt.subplots(figsize=(18,10));
feature.plot_matches(ax, image1, image2, keypoints1, keypoints2,
matches12);
plt.gray(); plt.show()
```



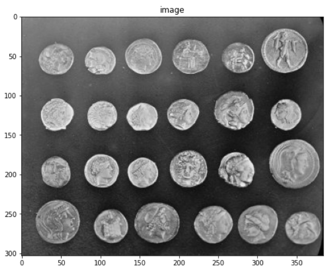

```
keypoints1 = feature.corner_peaks(feature.corner_harris(image1),
min_distance=5, threshold_rel=0.1)
keypoints2 = feature.corner_peaks(feature.corner_harris(image2),
min_distance=5, threshold_rel=0.1)

extractor = feature.BRIEF()

extractor.extract(image1, keypoints1)
keypoints1 = keypoints1[extractor.mask]
descr1     = extractor.descriptors
```

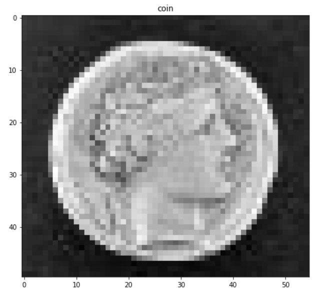
```
extractor.extract(image2, keypoints2)
keypoints2 = keypoints2[extractor.mask]
descr2 = extractor.descriptors


matches12 = feature.match_descriptors(descr1, descr2,
cross_check=True)

fig, ax = plt.subplots(figsize=(18,10));
feature.plot_matches(ax, image1, image2, keypoints1, keypoints2,
matches12);
plt.gray(); plt.show()
```

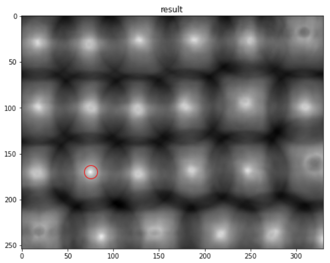


В некоторых случаях необходимо определить какой-либо объект на снимке. В этом случае skimage предоставляет методы сопоставления. Эти методы основаны на корреляции между шаблоном (объектом) и различными областями изображения.

```
image = skimage.data.coins()
coin = image[170:220, 75:130]
result = feature.match_template(image, coin)

plt.figure(figsize=(18,16))
plt.subplot(221),plt.imshow(image, 'gray');   plt.title('image')
plt.subplot(222),plt.imshow(coin, 'gray'); plt.title('coin')
plt.subplot(223),plt.imshow(result); plt.title('result')

y, x = np.unravel_index(np.argmax(result), result.shape)
plt.plot(x, y, 'o', markeredgecolor='r', markerfacecolor='none',
markersize=20)

plt.show()
```







Упражнение 2

3. Выберите два изображения и по крайней мере 5 методов работы с признками для них.
4. Сравните полученные признаки.

Лабораторная 3

Работа сопровождается документом в формате ipnb

Осоновы работы на PyTorch

Изучение особенностей библиотеки pytorch. Представление данных, методы работы с данными, представление изображений и их предобработка. Изучение полносвязного автоэнкодера для набора данных MNIST.

PyTorch - библиотека машинного обучения с открытым исходным кодом для Python. PyTorch используется для приложений машинного обучения и глубокие нейронные сети. Первоначально PyTorch был разработан исследовательской группой по искусственному интеллекту Facebook. В оличии от большенства других фреймворков машинного обучения PyTorch совместим с библиотеками numpy и scikit-learn.

Для установки pytorch необходимо зайти на офециальный сайт: https://pytorch.org/ в разделе INSTALL PYTORCH выбрать нужные опции.

* Для установки без видеоускорителя (для расчетов на CPU выбрать в разделе CUDA None)

Импорт torch

```
import torch
import numpy as np
import torch.nn as nn
from   torch.autograd import Variable
import torch.nn.functional as F

try:
   import summary
except:
   !pip install torchsummary
finally:
   from torchsummary import summary

import numpy as np
import matplotlib.pyplot as plt
```

настроим работу с torch выбирем формат работы и устройство для работы

```
dtype  = torch.float
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
print(device)
# device = torch.device("cuda:0") # Uncomment this to run on GPU
```

```
cpu
```

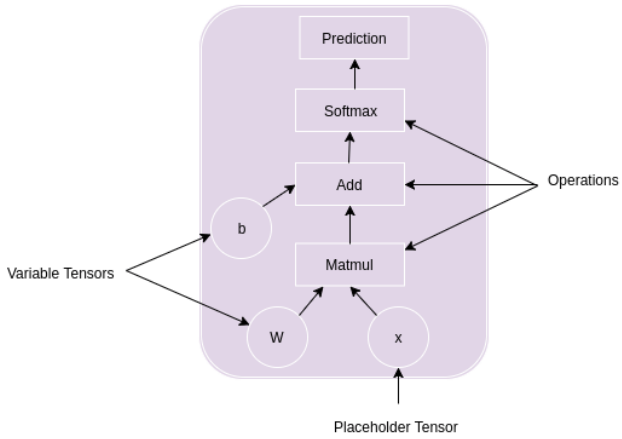
Для подробного изучения особенностей pytorch помимо данной работы разработчиками предлагуется начальный мануал с примерами;

статьи с описанием обучающих примеров;

репозиторий на GitHub с обучающими примерами.

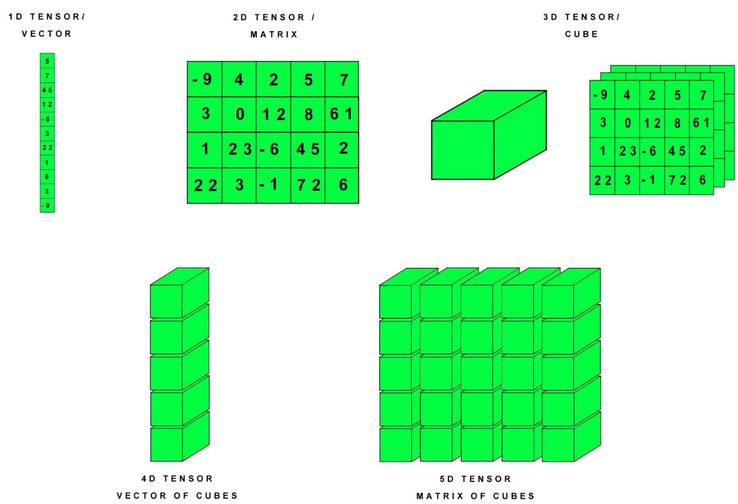
Типы данных

В PyTorch луюбая операция представляет собой граф вычислений. Каждый узел в графе представляет собой одну элементарную операцию. Пример такого графа представлен на рисунке ниже



Computational Graph

Основной тип данных в PyTorch - это тензор. Тензор в данном понимании отличается от стандартного, принятого в физике, и обозначает многомерную матрицу.



таким образом тензорами могут быть скаляры, векторы, матрицы и наборы матриц в нейронных сетях как правило наборы матриц трех-мерные.

Скаляр

```
x = torch.tensor(3.)
print(x)
print(x.size(), 'нулевая размерность')
```

tensor(3.)
torch.Size([]) нулевая размерность

Вектор

```
temp = torch.FloatTensor([1,2,3.4,-0.3])
temp.size()
print(temp)
print(temp.size())
```

```
tensor([ 1.0000,  2.0000,  3.4000, -0.3000])
torch.Size([4])
```

Матрица

попробуем задать из numpy

```
temp = torch.from_numpy(np.eye(4))
print(temp)
print(temp.size())
```

```
tensor([[1., 0., 0., 0.],
        [0., 1., 0., 0.],
        [0., 0., 1., 0.],
        [0., 0., 0., 1.]], dtype=torch.float64)
torch.Size([4, 4])
```

с матрицами и векторами в torch можно работать также как и в numpy

```
temp = temp[:2,:3]
print(temp)
print(temp.size())
```

```
tensor([[1., 0., 0.],
        [0., 1., 0.]], dtype=torch.float64)
torch.Size([2, 3])
```

Тензор

```
import torch
x = torch.Tensor(2, 3)
print(x)
print(x.shape)
```

```
tensor([[0.0000e+00, 1.8750e+00, 8.4078e-45],
        [0.0000e+00, 1.4013e-45, 0.0000e+00]])
torch.Size([2, 3])
```

Случайные тензор

```
x = torch.rand(2, 3)
print(x)

x = torch.rand(2, 3,2)
```

```
print(x)
print(x.shape,'трех-мерный тензор')
```

```
tensor([[0.0812, 0.6494, 0.3877],
        [0.4282, 0.8209, 0.3250]])
tensor([[[0.6844, 0.3428],
         [0.8327, 0.2470],
         [0.1784, 0.4802]],

        [[0.6151, 0.3245],
         [0.9564, 0.5970],
         [0.0632, 0.7664]]])
torch.Size([2, 3, 2]) трех-мерный тензор
```

Примеры операций с тензорами

С тензорами можно производить большенство стандартных операций, имеющихся в пакете NumPy

```
x = torch.ones(2,3)
y = torch.ones(2,3) * 2
c = torch.ones(2,3) * 0.1
z = x + y*c
print(z)
```

```
tensor([[1.2000, 1.2000, 1.2000],
        [1.2000, 1.2000, 1.2000]])
```

```
a = y[:,:1] + 1
print(a)
```

```
tensor([[3.],
        [3.]])
```

```
s = torch.sum(a)
print(s)
```

```
tensor(6.)
```

```
s = torch.pow(a,2)
print(s)
```

```
tensor([[9.],
        [9.]])
```

```
x = torch.randn(2, 3)
print(x)
s = torch.transpose(x,0,1)
print(s)
```

```
tensor([[-0.3215, -0.7384,  1.2512],
        [ 0.3673, -1.1259,  0.2302]])
tensor([[-0.3215,  0.3673],
```

```
        [-0.7384, -1.1259],
        [ 1.2512,  0.2302]]])
```

```
t = torch.tensor([1., 2.])
torch.save(t, 'tensor4test.pt')

del(t)

try:
   print(t)
except:
   print('now tensor does not exist')

t_new = torch.load('tensor4test.pt')
print('new t:',t_new)
```

```
now tensor does not exist
new t: tensor([1., 2.])
```

Переменные в PyTorch

В библиотеке PyTorch имеется ряд встроенных функций, которые могут быть автотмаически вызваны в коде. Одной из основных такхи функций является автоматизированный градиент. Данный механизм особенно важен при обучении нейронных сетей методом обратного распространения ошибки. Для осуществления автоматизированной работы со встроенными функциями в PyTorch используется тип данных переменные.

Создадим переменную из простого тензора, для которой необходим градиент (она может быть использована для обучения):

```
x = Variable(torch.ones(2, 2) * 2, requires_grad=True)
print(x)
```

```
tensor([[2., 2.],
        [2., 2.]], requires_grad=True)
```

Тензор тоже может быть переменным.

```
x = torch.tensor([3.,4.])
w = torch.tensor([4.,8], requires_grad=True)
b = torch.tensor(5., requires_grad=True)
print(x,y,b)
```

с переменными можно выполнять теже действия, что и с тензорами

```
y = w * x + b
print(y)
```

```
tensor([17., 37.], grad_fn=<AddBackward0>)
```

Однако, с переменными можно задать и специфическую операцию - получение градиента

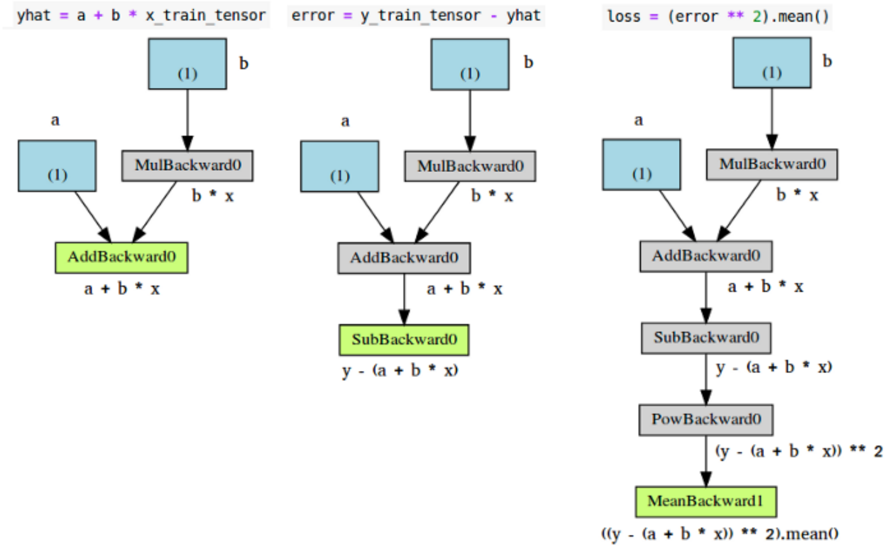
! обратите внимание, что обратное распростронение ошибки работает только для скаляра!

*y = w * x + b*
dy = torch.sum(y).backward()

print(''' sum of dy/dw:''', w.grad)
print('sum of dy/db:', b.grad)

sum of dy/dw: tensor([3., 4.])
sum of dy/db: tensor(2.)

PyTorch собирает все этапы вычислений в единый граф. Данный граф формируется динамически для каждой сессии. Работу данного графа можно проиллюстрировать следующим образом для расчет функции потерь



Когда граф построен, он может быть выполнен или на CPU или на GPU Nvidia (с использованием фреймворка cuda).

Встроенные функции активации в PyTorch

data = torch.randn(2, 2)
print(data)
print(torch.relu(data))

data = torch.randn(2, 2)
print(data)
print(torch.sigmoid(data))

data = torch.randn(2, 2)
print(data)
print(torch.tanh(data))

Встроенные слои в PyTorch

Applies a linear transformation to the incoming data: :math:`y = xA^T + b`

```
lin = nn.Linear(5, 3)  # maps from R^5 to R^3, parameters A, b
# data is 2x5.  A maps from 5 to 3... can we map "data" under A?
data = torch.randn(2, 5)
print(lin(data))
print(lin(data).size())

tensor([[-0.5038, -0.5178,  0.3489],
        [ 0.1732, -1.0476, -0.0381]], grad_fn=<AddmmBackward>)
torch.Size([2, 3])

# Applies a 1D convolution over an input signal composed of several input planes.

lin = nn.Conv1d(in_channels  = 5,
          out_channels = 3,
          kernel_size  = 2,
          stride       = 1,
          padding_mode = 'zeros',)
# data is 2x5.  A maps from 5 to 3... can we map "data" under A?
data = torch.randn(1, 5, 2)
print(lin(data))
print(lin(data).size())

tensor([[[-0.0428],
         [-0.0871],
         [ 0.4935]]], grad_fn=<SqueezeBackward1>)
torch.Size([1, 3, 1])

# Applies a 1D max pooling over an input signal composed of several input planes.

lin = nn.MaxPool1d(kernel_size = 2,
           stride      = 1)
# data is 2x5.  A maps from 5 to 3... can we map "data" under A?
data = torch.randn(1, 5, 2)
print(lin(data))
print(lin(data).size())

tensor([[[-0.6559],
         [ 1.5549],
         [ 0.1622],
         [-0.6880],
         [ 0.1080]]])
torch.Size([1, 5, 1])

# Applies a 2D convolution over an input signal composed of several input planes.

lin = nn.Conv2d(in_channels  = 5,
          out_channels = 3,
          kernel_size  = 2,
          stride       = 1,
          padding_mode = 'zeros',)

data = torch.randn(1, 5, 4, 2)
```

```
print(lin(data))
print(lin(data).size())
```

```
tensor([[[[-0.1818],
          [-0.2898],
          [ 0.6576]],

         [[ 0.6750],
          [-1.0868],
          [-0.7502]],

         [[-0.3926],
          [-0.7923],
          [ 0.0696]]]], grad_fn=<ThnnConv2DBackward>)
torch.Size([1, 3, 3, 1])
```

Applies a multi-layer Elman RNN with :math:`tanh` or :math:`ReLU` non-linearity to an input sequence.

```
rnn  = nn.RNN(3, 2, 2)
data = torch.randn(3, 2, 3)
h0   = torch.randn(2, 2, 2)
c0   = torch.randn(2, 2, 2)
output, hn = rnn(data, h0)
print(output)
print(output.size())
```

```
tensor([[[-0.9743, -0.3390],
         [-0.6847, -0.3865]],

        [[-0.6346,  0.3954],
         [-0.4316, -0.0528]],

        [[-0.7212, -0.0350],
         [-0.9085,  0.6949]]], grad_fn=<StackBackward>)
torch.Size([3, 2, 2])
```

Applies a multi-layer long short-term memory (LSTM) RNN to an input sequence.

```
rnn  = nn.LSTM(3, 2, 2)
data = torch.randn(2, 2, 3)
h0   = torch.randn(2, 2, 2)
c0   = torch.randn(2, 2, 2)
output, (hn, cn) = rnn(data, (h0, c0))
print(output)
print(output.size())
```

```
tensor([[[ 0.3315, -0.1456],
         [ 0.1554,  0.0448]],

        [[ 0.0270,  0.0837],
```

```
        [-0.0015,  0.1473]]], grad_fn=<StackBackward>)
torch.Size([2, 2, 2])
```

Линейная регрессия на PyTorch

Выражение для регресси

$y=a\cdotx+b$

где:

Значение a является наклоном.

Значение b — это y — пересечение.

r — коэффициент корреляции.

r^2— коэффициент корреляции.

В более общем - матричном виде

$$y = X \cdot W^T + b$$

где:

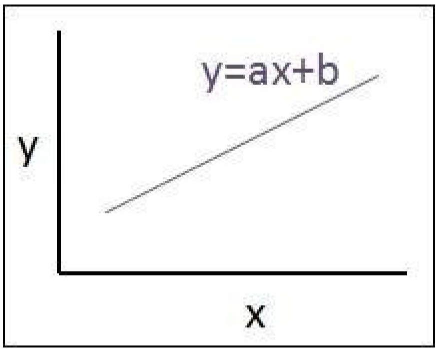
X - это скаляр/матрица/тензор входных значений;

W - это скаляр/матрица/тензор весов (в случае скаляра W = a);

b - это скаляр/вектор/матрица смещений (на размер меньше, чем W);

y - это скаляр/матрица/тензор выходных значений.

Отметим, что часто набор W включает в себя b в качестве нулевой или последней составляющей.



Реализуем регрессию в pyTorch в ручном режиме

импорт основных библиотек

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
import torch
import torch.nn as nn
from torch.autograd import Variable
```

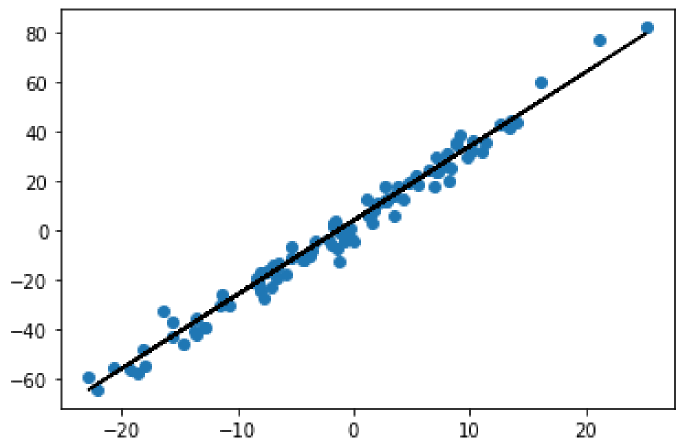
```
%matplotlib inline

torch.manual_seed(1969)

# import seaborn as sns
# import pandas as pd
# sns.set_style(style = 'whitegrid')
# plt.rcParams["patch.force_edgecolor"] = True
```

<torch._C.Generator at 0x1add33ddc30>

создадим датасет

```
x = torch.randn(100, 1) * 10
y_clear = 3*x + 4
y = y_clear + 5 * torch.randn(100, 1)

plt.plot(x,y,'o')
plt.plot(x,y_clear,'-k')
```

[<matplotlib.lines.Line2D at 0x1adddb5fcc8>]




```
# x = np.atleast_2d(x)
# y = np.atleast_2d(y)

# inputs  = torch.from_numpy(x)
# targets = torch.from_numpy(y)
inputs = x
targets = y
print(inputs.shape)
```

torch.Size([100, 1])

зададим веса и смещения случайными тензорами

```
w = torch.randn(1, requires_grad=True)
b = torch.randn(1, requires_grad=True)
print(w)
print(b)
```

tensor([0.7502], requires_grad=True)
tensor([-0.4575], requires_grad=True)

Определим модель линейной регрессии

```
def model(x,w,b):
  return x * w + b
```

Зададим функцию потерь

$$MSE = \sum(y - x)^2/N$$

```
def mse(predicts, targets):
  diff = torch.abs(predicts - targets)
  return torch.sum(diff * diff) / diff.numel()
```

Посмотрим на начальные значения предсказаний и сравним их с ожидаемыми значениями путем подчета функции потерь (MSE)

```
predicts = model(inputs,w,b)
# print(predicts)

loss = mse(predicts, targets)
print('loss = ',loss.data.numpy())
```

loss = 515.2697

Пока значение функции потерь большое модель будет давать большую ошибку предсказания

Для снижения функции потерь будем использовать метод градиентного спукска.

Посчитаем один шаг такого градиента

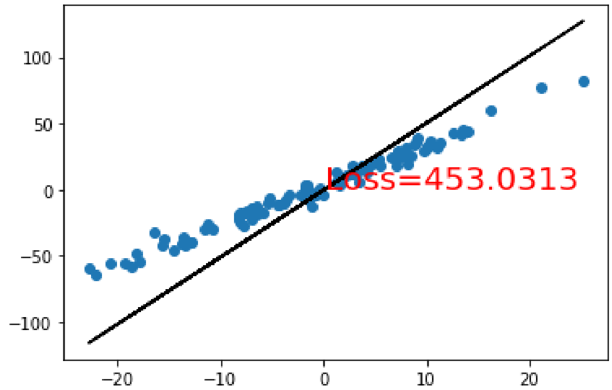
```
predicts = model(inputs,w,b)
loss = mse(predicts, targets)
loss.backward()
db = b.grad
dw = w.grad
```

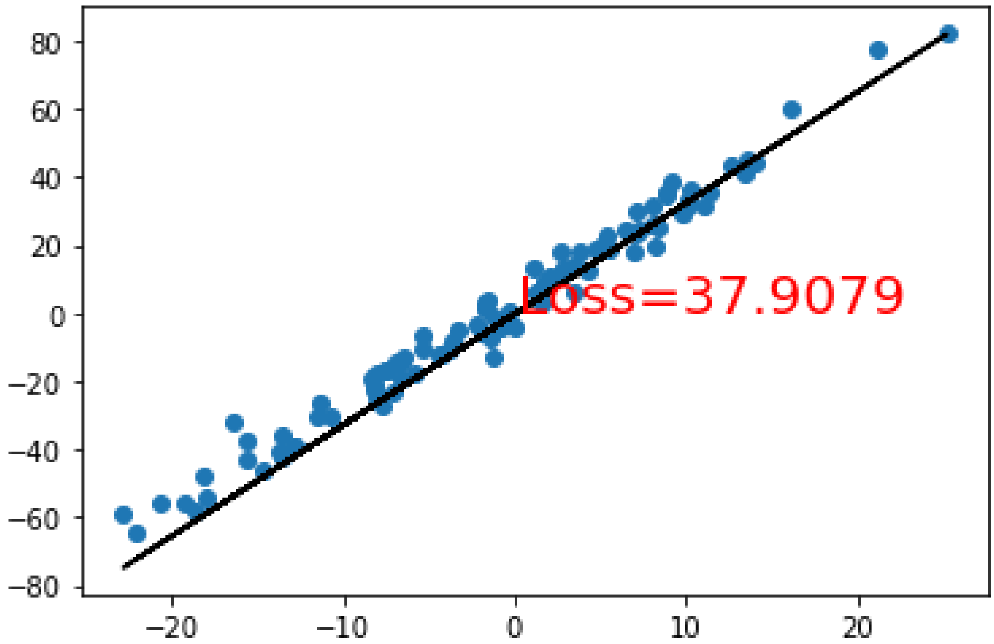
Теперь попробуем обновить веса и сброситьградиент

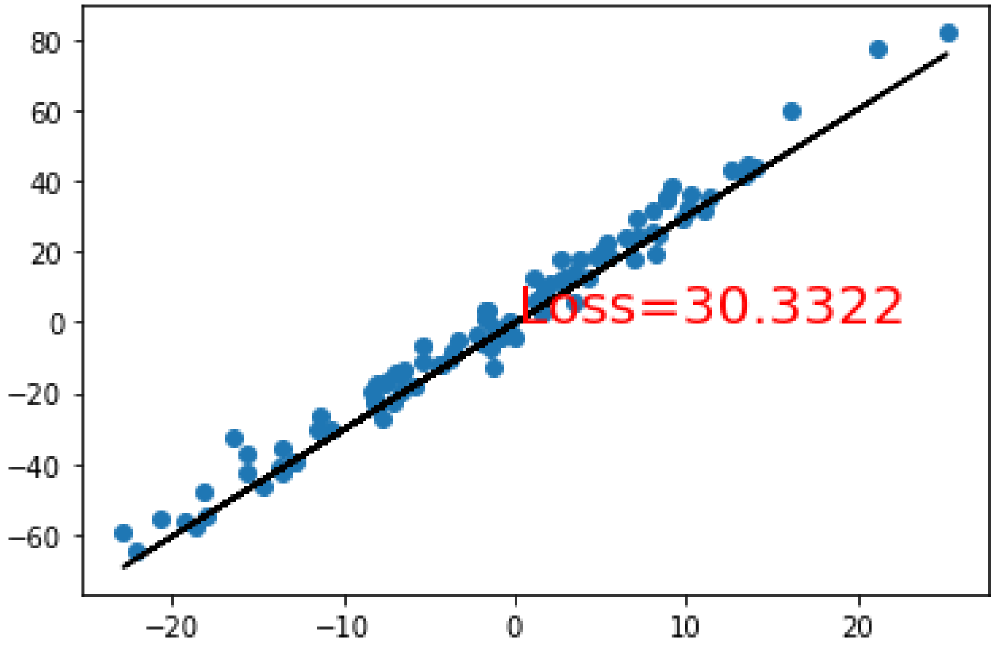
```
LR = 0.01

# Adjust weights & reset gradients
with torch.no_grad():
  w -= w.grad * LR
  b -= b.grad * LR
  w.grad.zero_()
  b.grad.zero_()
```

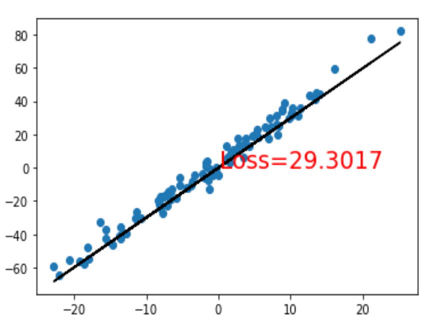
Проведем данную процедуру иттеративно, например 100 эпох

```
epochs = 100
LR = 0.0005

trainig = np.zeros(epochs)
for i in range(epochs):
    predicts = model(inputs,w,b)
    loss = mse(predicts, targets)
    trainig[i] = loss.data.numpy()
    loss.backward()
    with torch.no_grad():
        w -= w.grad * LR
        b -= b.grad * LR
        w.grad.zero_()
        b.grad.zero_()

    if(i%20 == 0):
        plt.plot(x,targets.data.numpy(),'o')
        plt.plot(x,predicts.data.numpy(),'-k')
        plt.text(0.1, 0.1, 'Loss=%.4f' %(trainig[i]), fontdict={'size': 20, 'color': 'red'})
        plt.show()

plt.plot(trainig)
```









[<matplotlib.lines.Line2D at 0x1adddda396c8>]

Посчитаем, что получилось.

```
predicts = model(inputs,w,b)
loss = mse(predicts, targets)
print(loss)
```

```
tensor(28.8742, grad_fn=<DivBackward0>)
```

Полученные значения веса (наклон) и смещения

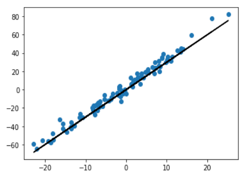
```
print(w)
print(b)
```

```
tensor([2.9912], requires_grad=True)
tensor([-0.0708], requires_grad=True)
```

результат

```
plt.plot(x,targets.data.numpy(),'o')
plt.plot(x,predicts.data.numpy(),'-k')
```

```
[<matplotlib.lines.Line2D at 0x1addd910e08>]
```



Реализуем регрессию в PyTorch при помощи встроенных методов

теперь сделаем регрессию в более поплуярном в PyTorch виде

Для этого запишем класс линейная регрессия.

!В PyTorch каждый класс модели желательно, чтобы наследовал от базового класса nn.Module!

```
class LinearRegression(nn.Module):
  def __init__(self, in_features=1, out_features=1 ):
    super().__init__()
    self.l1 = nn.Linear(in_features=in_features, out_features=out_features)
  def forward(self, x):
    return self.l1(x)
```

Создадим модель - экземпляр нашего класса

```
model = LinearRegression(1,1)
model
```

```
LinearRegression(
  (l1): Linear(in_features=1, out_features=1, bias=True)
)
```

Теперь выберем стандартную функцию потерь и оптимизатор (метод градиентного спуска)

```
LR = 0.01
criterion = nn.MSELoss()
optimizer = torch.optim.SGD(model.parameters(), lr=LR)

w,b   = model.parameters()
print(w,b)
```

```
Parameter containing:
tensor([[-0.7876]], requires_grad=True) Parameter containing:
tensor([0.6797], requires_grad=True)
```

```
plt.plot(x,y,'o')
plt.plot(x,y_clear,'-k')
#Предыдущие данные
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-9-fd93063482bb> in <module>
----> 1 plt.plot(x,y,'o')
      2 plt.plot(x,y_clear,'-k')
      3 #Предыдущие данные

NameError: name 'x' is not defined
```

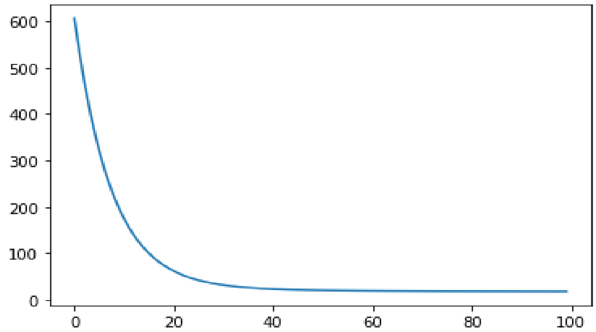
Проведем обучение сети

```
epochs = 100
losses = np.zeros(epochs)

for i in range(epochs):
    predict = model.forward(x)

    loss = criterion(predict, y)

    losses[i] = loss.data.numpy()

    optimizer.zero_grad()

    loss.backward()

    optimizer.step()

    if(i//20 == i/20):
        print("Epoch: ", i+1, " Loss: ", loss.item())

plt.plot(losses)
```

```
Epoch:  1  Loss:  605.1109619140625
Epoch:  21  Loss:  62.147361755371094
```

Epoch: 41 Loss: 23.663070678710938
Epoch: 61 Loss: 19.759572982788086
Epoch: 81 Loss: 18.87649917602539

[<matplotlib.lines.Line2D at 0x1addd3befc8>]

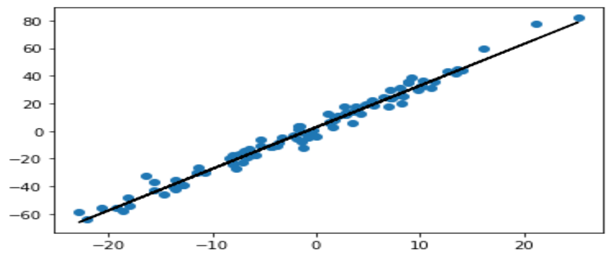


```
[w, b]    = model.parameters()
print(w[0,0],b[0])
```

tensor(3.0209, grad_fn=<SelectBackward>) tensor(2.6837, grad_fn=<SelectBackward>)

```
predict = model.forward(x)
plt.plot(x,y.data.numpy(),'o')
plt.plot(x,predict.data.numpy(),'-k')
```

[<matplotlib.lines.Line2D at 0x1adddcfa848>]



Посмотрим на то, какие данные из модели можно получать. Для этого есть несколько способов

```
print(list(model.parameters()))
print(list(model.state_dict()))
print(list(model.named_buffers()))
print(list(model.named_parameters()))
```

```
[Parameter containing:
tensor([[3.0209]], requires_grad=True), Parameter containing:
tensor([2.6837], requires_grad=True)]
['l1.weight', 'l1.bias']
[]
[('l1.weight', Parameter containing:
tensor([[3.0209]], requires_grad=True)), ('l1.bias', Parameter containing:
tensor([2.6837], requires_grad=True))]
```

```
num_params = sum(p.numel() for p in model.parameters() if p.requires_grad)
print('Number of trainable parameters for the model: %d' % (num_params))
```

```
num_params = sum(p.numel() for p in model.parameters() )
print('Number of all parameters for the model: %d' % (num_params))
```

```
Number of trainable parameters for the model: 2
Number of all parameters for the model: 2
```

Также для оценки модели есть специальная функция *summary* из библиотеки *torchsummary*

```
summary(model,input_size = x.shape[1:])
```

```
Input size (MB): 0.00
Forward/backward pass size (MB): 0.00
Params size (MB): 0.00
Estimated Total Size (MB): 0.00
----------------------------------------------------------------
```

теперь попробуем сохранить модель

```
torch.save(model.state_dict(), 'test_module.pt')
```

удалим модель

```
del(model)
```

теперь заново загруим модель, но назовем ее *new_model*

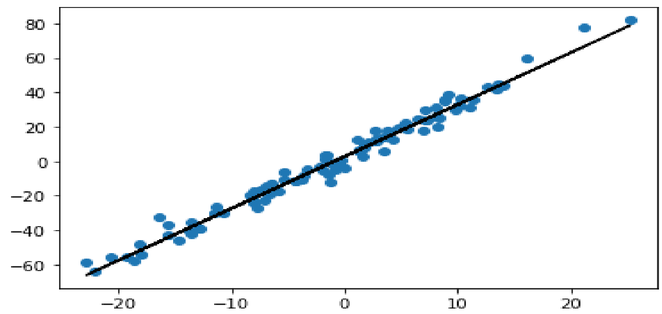
```
new_state_dict = torch.load('test_module.pt')
new_model = LinearRegression(1,1)
new_model.load_state_dict(new_state_dict)
```

```
<All keys matched successfully>
```

проверим, что новая модель соответствует обученной прежде

```
predict = new_model.forward(x)
plt.plot(x,y.data.numpy(),'o')
plt.plot(x,predict.data.numpy(),'-k')
```

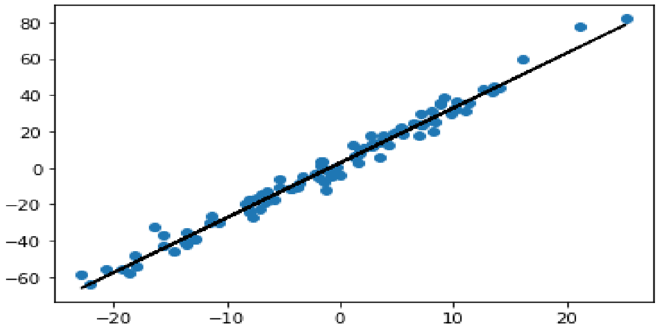
```
[<matplotlib.lines.Line2D at 0x1addd2d9808>]
```



Также модель можно сохранить с использованием JIT скрипотов, что полезно для ее последующего портирования, например на C++

```
scripted_module = torch.jit.script(new_model)
torch.jit.save(scripted_module, 'mymodule.pt')
again_new_model = torch.jit.load('mymodule.pt')

predict = again_new_model.forward(x)
plt.plot(x,y.data.numpy(),'o')
plt.plot(x,predict.data.numpy(),'-k')
```

[<matplotlib.lines.Line2D at 0x1f9a0597b08>]



Смотрите больше примеров работы с моделями тут.

Упражнение 1

реализуйте квадратичную регрессию в при помощи встроенных методов PyTorch

$y = w_1 \cdot x^2 + w_2 \cdot x + b = w_1 \cdot x_1 + w_2 \cdot x_2 + b$

Упражнение 2

В место вручную описанного класс LinearRegression можно использовать стандрартные шаблоны для создания нейронных сетей.

Одним из таких шаблонов является `Sequential`
вместе с данным шаблоном модель можно переписать в следующем виде

```
model = nn.Sequential(nn.Linear(n_inputs, n_outputs))
```

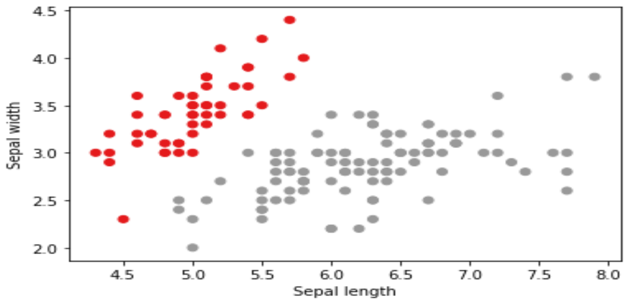
Задание: Перепешите Описанный выше пример с использованием шаблона `Sequential`

Логистическая регрессия на PyTorch

Создадаим датасет ирисов с двумя классами, входные данные сделаем дву-мерными

```
from sklearn import datasets
from sklearn.linear_model import LogisticRegression
import numpy as np
from sklearn.model_selection  import train_test_split

iris = datasets.load_iris()

x = iris.data[:, :2]
```

```python
y = (iris.target != 0)

# Plot also the training points
plt.scatter(x[:, 0], x[:, 1], c=y, cmap=plt.cm.Set1)
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')

plt.show()
```



выделим тренировочную и тестовую части

```python
x_train, x_test, y_train, y_test = train_test_split(x, y,
test_size=0.3, random_state=0)

x_train = torch.from_numpy(x_train.astype(np.float32))
x_test  = torch.from_numpy(x_test.astype(np.float32))
y_train = torch.from_numpy(y_train.astype(np.float32).reshape(-1,1))
y_test  = torch.from_numpy(y_test.astype(np.float32).reshape(-1,1))

print(x_train.shape, x_test.shape, y_train.shape, y_test.shape)

torch.Size([105, 2]) torch.Size([45, 2]) torch.Size([105, 1])
torch.Size([45, 1])
```

создадим класс регрессора, аналогично, как в случае линейной регрессии

```python
class LogisticRegression(nn.Module):

    def __init__(self, in_features=2, n_classes=1 ):
        super().__init__()
        self.l1 = nn.Linear(in_features=in_features,
out_features=n_classes)
    def forward(self, x):
        return torch.sigmoid(self.l1(x))
```

опишим параметры обучения

- **Будем использовать в качестве функции потерь бинарную энтрпию**
- **Обучение проведем при помощи стохастического градиентного спуска**

```python
LR = 0.1

model = LogisticRegression(x_train.shape[1],y_train.shape[1])

criterion = nn.BCELoss()
```

```
optimizer = torch.optim.SGD(model.parameters(), lr=LR)

w,b   = model.parameters()
print(w,b)

summary(model,input_size = x.shape[1:])
```

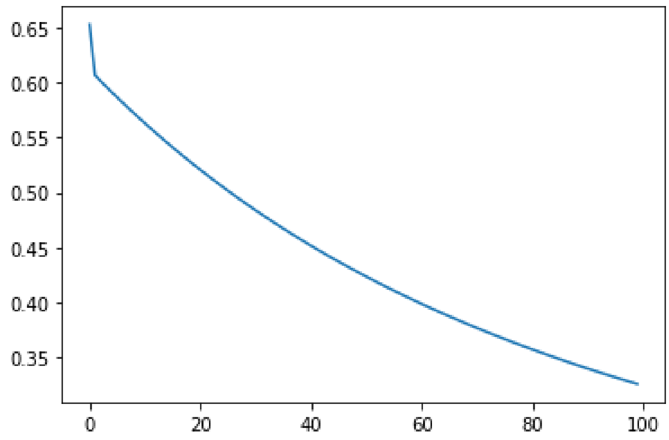
```
Parameter containing:
tensor([[0.3340, 0.3726]], requires_grad=True) Parameter containing:
tensor([0.3685], requires_grad=True)
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1                    [-1, 1]               3
================================================================
Total params: 3
Trainable params: 3
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.00
Params size (MB): 0.00
Estimated Total Size (MB): 0.00
----------------------------------------------------------------
```

Проведем процесс тренировки регрессора

```
epochs = 100
losses = np.zeros(epochs)

for i in range(epochs):
    predict = model.forward(x_train)

    loss = criterion(predict, y_train)

    losses[i] = loss.data.numpy()

    optimizer.zero_grad()

    loss.backward()

    optimizer.step()

    if(i//20 == i/20):
        print("Epoch: ", i+1, " Loss: ", loss.item())

plt.plot(losses)
```

```
Epoch:  1  Loss:  0.6525978446006775
Epoch:  21  Loss:  0.520682692527771
Epoch:  41  Loss:  0.4516817033290863
Epoch:  61  Loss:  0.3987092971801758
Epoch:  81  Loss:  0.35733699798583984
```

```
[<matplotlib.lines.Line2D at 0x1f9a2d1d6c8>]
```



Посмотрим на ошибку

```
predict = model.forward(x_test)

print(torch.mean(torch.round(predict)-torch.round(y_test)).data)

tensor(0.0222)
```

Попытаемся ее визуализировать и сравнить ожидаемые значения ('Ground Truth') и полученные ('Predicted')

```
plt.figure(figsize=(16,10))

plt.subplot(221)
# Plot also the training points
plt.scatter(x_test[:, 0], x_test[:, 1], c=torch.squeeze(y_test),
cmap=plt.cm.Set1)
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')
plt.title('Ground Truth FOR TEST')



predict = model.forward(x_test)
lables_predicted = predict.data.numpy().argmax(axis=1)

plt.subplot(222)
# Plot also the training points
plt.scatter(x_test[:, 0], x_test[:, 1], c=lables_predicted,
cmap=plt.cm.Set1)
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')
plt.title('Predicted FOR TEST')


predict = model.forward(torch.from_numpy(x.astype(np.float32)))
lables_predicted = predict.data.numpy().argmax(axis=1)

plt.subplot(223)
```

```
# Plot also the training points
plt.scatter(x[:, 0], x[:, 1], c=np.round(np.squeeze(y)),
cmap=plt.cm.Set1)
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')
plt.title('Ground Truth FOR ALL')

plt.subplot(224)
# Plot also the training points
plt.scatter(x[:, 0], x[:, 1], c=lables_predicted, cmap=plt.cm.Set1)
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')
plt.title('Predicted FOR ALL')
```

```
----------------------------------------------------------------------
-----
NameError                                 Traceback (most recent call
last)
<ipython-input-2-e45b08a05c12> in <module>
----> 1 plt.figure(figsize=(16,10))
      2
      3 plt.subplot(221)
      4 # Plot also the training points
      5 plt.scatter(x_test[:, 0], x_test[:, 1],
c=torch.squeeze(y_test), cmap=plt.cm.Set1)

NameError: name 'plt' is not defined
```

Упражнение 1

проверьте как на качество обучение повлияет снижение скорости обучения, как увеличение повлияет снижение эпох обучения

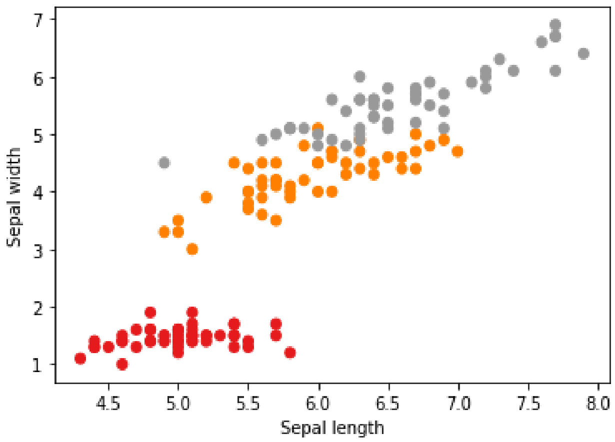
Много-классовая логистическая регрессия

Рассмотрим особенности замены бинарной логистической регрессии на много-классовую (Софт-макс) регрессию

Датасет из 3 классов

```
from sklearn import datasets
import matplotlib.pyplot as plt
import numpy as np

iris = datasets.load_iris()

x = iris.data[:,[0,2] ]
y = (iris.target)

plt.scatter(x[:, 0], x[:, 1], c=y, cmap=plt.cm.Set1)
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')
```

```
plt.show()

# standardize
x[:,0] = (x[:,0] - x[:,0].mean()) / x[:,0].std()
x[:,1] = (x[:,1] - x[:,1].mean()) / x[:,1].std()

print(x.shape)
print(y.shape)


x_train, x_test, y_train, y_test = train_test_split(x, y,
test_size=0.3, random_state=0)

x_train = torch.from_numpy(x_train.astype(np.float32))
x_test  = torch.from_numpy(x_test.astype(np.float32))
y_train = torch.from_numpy(y_train.astype(np.float32).reshape(-1,1))
y_test  = torch.from_numpy(y_test.astype(np.float32).reshape(-1,1))

print(x_train.shape, x_test.shape, y_train.shape, y_test.shape)
```




```
(150, 2)
(150,)
torch.Size([105, 2]) torch.Size([45, 2]) torch.Size([105, 1])
torch.Size([45, 1])
```

Класс для регрессии

```
class SoftmaxRegression(LogisticRegression):

    def __init__(self,
                 in_features = 2,
                 n_classes   = 3):

        super().__init__(in_features = in_features,
                         n_classes   = n_classes)

    def forward(self, x):
        return torch.nn.functional.softmax(self.l1(x), dim=1)
```

Обучение при помощи функции Взаимной энтропии

```
LR = 0.1

model = SoftmaxRegression(x_train.shape[1],3)

criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr=LR)

w,b   = model.parameters()
print(w,b)
summary(model,input_size = x.shape[1:])
```

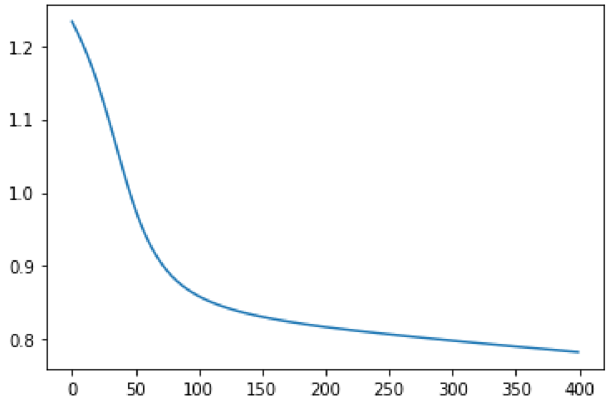
```
Parameter containing:
tensor([[-0.4953, -0.4105],
        [-0.2640,  0.0957],
        [-0.4164,  0.1461]], requires_grad=True) Parameter containing:
tensor([-0.2691, -0.0484, -0.2824], requires_grad=True)
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1                    [-1, 3]               9
================================================================
Total params: 9
Trainable params: 9
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.00
Params size (MB): 0.00
Estimated Total Size (MB): 0.00
----------------------------------------------------------------
```

Процесс обучения

```
epochs = 400
losses = np.zeros(epochs)

for i in range(epochs):

    predict = model.forward(x_train)

    loss = criterion(predict, torch.squeeze(y_train).type(torch.long))

    losses[i] = loss.data.numpy()

    optimizer.zero_grad()

    loss.backward()

    optimizer.step()

    if(i//(epochs//5) == i/(epochs//5)):
```

```
        print("Epoch: ", i+1, " Loss: ", loss.item())

plt.plot(losses)
```

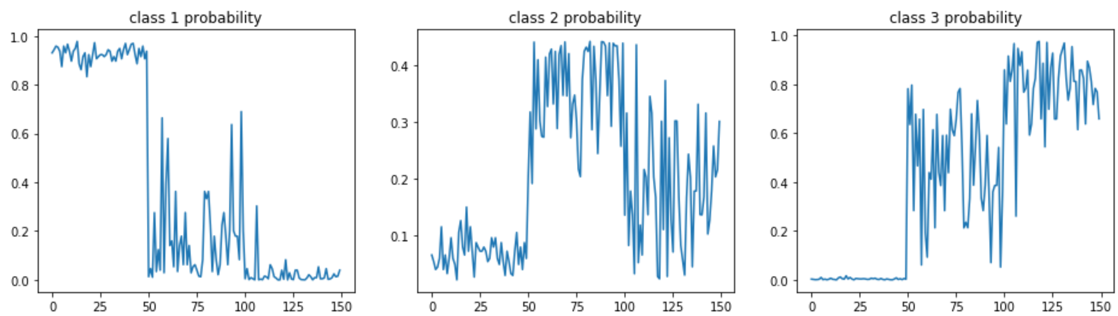
```
Epoch:  1  Loss:  1.2328581809997559
Epoch:  81  Loss:  0.8841432332992554
Epoch:  161  Loss:  0.8274716138839722
Epoch:  241  Loss:  0.8089736700057983
Epoch:  321  Loss:  0.7952234148979187

[<matplotlib.lines.Line2D at 0x1f9a2fb5f88>]
```



В результате функция предсказания выдает 3 класса в виде массива

```
predict = model.forward(torch.from_numpy(x).type(torch.float))
print(predict.shape)
plt.figure(figsize = (16,4))
plt.subplot(1,3,1);plt.plot(predict.data.numpy()[:,0]);plt.title('class 1 probability')
plt.subplot(1,3,2);plt.plot(predict.data.numpy()[:,1]);plt.title('class 2 probability')
plt.subplot(1,3,3);plt.plot(predict.data.numpy()[:,2]);plt.title('class 3 probability')
plt.show()
```

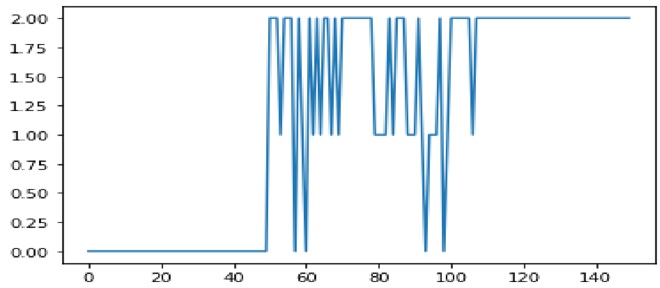
```
torch.Size([150, 3])
```



Приведем их к одному вектору

```
lables_predicted = predict.data.numpy().argmax(axis=1)
lables_predicted.shape
plt.plot(lables_predicted)
```

```
[<matplotlib.lines.Line2D at 0x1f9a2f6e488>]
```



Визуализация

```
plt.figure(figsize=(16,10))

plt.subplot(221)
# Plot also the training points
plt.scatter(x_test[:, 0], x_test[:, 1], c=torch.squeeze(y_test),
cmap=plt.cm.Set1)
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')
plt.title('Ground Truth FOR TEST')



predict = model.forward(x_test)
lables_predicted = predict.data.numpy().argmax(axis=1)

plt.subplot(222)
# Plot also the training points
plt.scatter(x_test[:, 0], x_test[:, 1], c=lables_predicted,
cmap=plt.cm.Set1)
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')
plt.title('Predicted FOR TEST')


predict = model.forward(torch.from_numpy(x.astype(np.float32)))
lables_predicted = predict.data.numpy().argmax(axis=1)

plt.subplot(223)
# Plot also the training points
plt.scatter(x[:, 0], x[:, 1], c=np.round(np.squeeze(y)),
cmap=plt.cm.Set1)
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')
plt.title('Ground Truth FOR ALL')

plt.subplot(224)
# Plot also the training points
plt.scatter(x[:, 0], x[:, 1], c=lables_predicted, cmap=plt.cm.Set1)
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')
plt.title('Predicted FOR ALL')
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-1-e45b08a05c12> in <module>
----> 1 plt.figure(figsize=(16,10))
      2 
      3 plt.subplot(221)
      4 # Plot also the training points
      5 plt.scatter(x_test[:, 0], x_test[:, 1], c=torch.squeeze(y_test), cmap=plt.cm.Set1)

NameError: name 'plt' is not defined
```

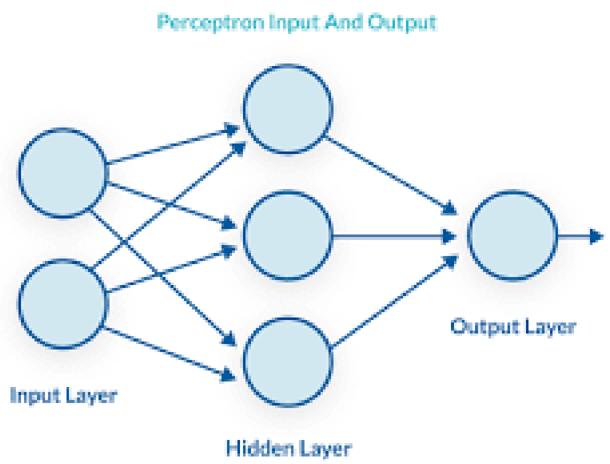
Упражнение 2

Перепишите модель логистической регрессии с использованием шаблона

Sequential

Полносвязная нейронная сесть на PyTorch

Давайте протестируем torch в задаче обучения полносвязной нейронной сети




```
import torch
import torchvision
from torchvision import transforms, datasets

import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
```

Мы будем использовать популярный набор данных MNIST.

Набор данных MNIST - это база данных рукописных цифр (0-9) - один из самых популярных наборов данных в компьютерном зрении. MNIST содержит 60 000 обучающих изображений и 10 000 тестовых изображений. Каждое изображение имеет размеры 28 на 28 пикселей в оттенках серого.
Мы будем загружать данные, используя библиотеку `torchvision`. Для преобрзования

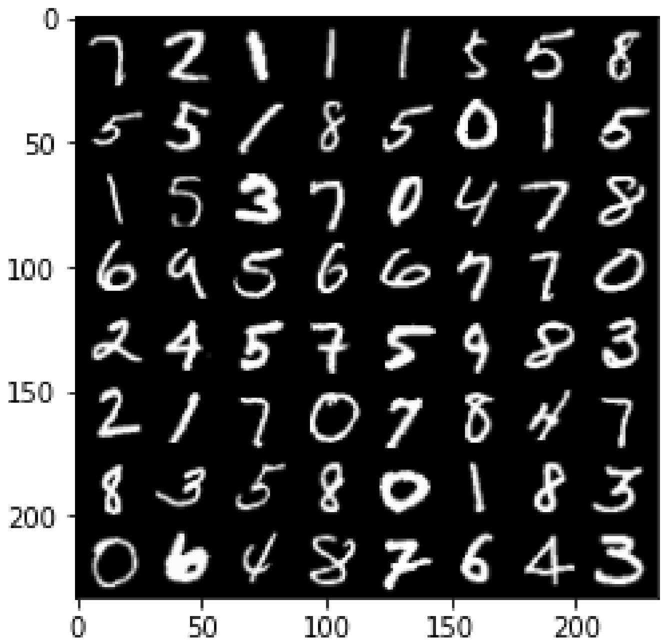
набора данных мы будем использовать преобразования `transforms.Compose`. В конце мы будем использовать функцию torch.utils.data.DataLoader () для подготовки загрузки данных для обучения и тестирования.

```
BATCH_SIZE = 64

transform=transforms.Compose([
                              transforms.ToTensor()
                               ])

train = torchvision.datasets.MNIST('',
                                   train=True,
                                   download=True,
                                   transform  = transform)

test  = torchvision.datasets.MNIST('',
                                   train=False,
                                   download=True,
                                   transform =transform)

trainset = torch.utils.data.DataLoader(train, batch_size=BATCH_SIZE,
shuffle=True)

testset  = torch.utils.data.DataLoader(test, batch_size=BATCH_SIZE,
shuffle=False)

trainset_shape = trainset.dataset.data.shape
testset_shape  = testset.dataset.data.shape

print(trainset_shape, testset_shape)
```

```
torch.Size([60000, 28, 28]) torch.Size([10000, 28, 28])
```

```
import matplotlib.pyplot as plt
import numpy as np

def image_show(images):
    images = images.numpy()
    images = images.transpose((1, 2, 0))
    print(images.shape)
    plt.imshow(images)
    plt.show()

# get some random training images batch
dataiter = next(iter(trainset))

# parse images and lables from batch
images, _ = dataiter

## show images
image_show(torchvision.utils.make_grid(images, nrow = 8, padding = 1))
```

```
(233, 233, 3)
```



обучение
Создаем класс Net для модели. Модель содержит 3 слоя. Каждый скрытый слой содержит 64 выхода. В выходном слое - 10 выходов, представляющих десять классов изображений. Для скрытых слоев мы будем использовать функцию активации `relu` (Rectified Linear Activation). Для выходного слоя мы будем использовать функцию активации `softmax`.

- Примечание * вместо просто $softmax$ мы будем использовать $log(softmax(x))$ из-за требований к реализации функции потерь.

```
IMAGE_WIDTH = 28
IMAGE_HEIGH = 28

class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(IMAGE_WIDTH*IMAGE_HEIGH, 64)
        self.fc2 = nn.Linear(64, 64)
        self.fc3 = nn.Linear(64, 10)

    def forward(self, x):
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return F.log_softmax(x, dim=1)   #log(softmax(x))


net = Net()
# print(net)
summary(net,input_size =(1,784))
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1                [-1, 1, 64]          50,240
            Linear-2                [-1, 1, 64]           4,160
```

```
            Linear-3                    [-1, 1, 10]             650
================================================================
Total params: 55,050
Trainable params: 55,050
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.00
Params size (MB): 0.21
Estimated Total Size (MB): 0.21
----------------------------------------------------------------
```

Мы будем использовать оптимизатор *ADAM* для оптимизации сети. Мы будем использовать кросс-энтропию как функцию потерь. Аргумент `lr` определяет скорость обучения функции оптимизатора.

```
criterion = nn.CrossEntropyLoss() #This criterion requires `log softmax`
optimizer = optim.Adam(net.parameters(), lr=0.005)
```

Обучение.

У нас будет пять полных проходов по данным (5 эпох).
Функция `net.zero_grad()` устанавливает нулевые градиенты перед вычислением убытков. Функция `net(X.view (-1,784))` позволяет преобразовать изображения в одномерные векторы. Число *784* является результатом преобразования изображения с размерами 28 на 28.
Функция `criterion(output, y)` - это функция потерь.
Последняя строка кода выводит потери для каждой эпохи.

```
EPOCHS = 5

for epoch in range(EPOCHS):
    for data in trainset:
        X, y = data
        net.zero_grad()
        output = net(X.view(-1,784))
        loss = criterion(output, y)
        loss.backward()
        optimizer.step()
    print('[epoch:%d/%d]'%(epoch,EPOCHS), 'loss val = 
%.4f'%(loss.item()) )
```

```
[epoch:0/5] loss val = 0.0801
[epoch:1/5] loss val = 0.2674
[epoch:2/5] loss val = 0.1181
[epoch:3/5] loss val = 0.0115
[epoch:4/5] loss val = 0.0114
```

Проверка

```
correct = 0
total   = 0
```

```
with torch.no_grad():
    for data in testset:
        X, y = data
        output = net(X.view(-1,784))

        for idx, i in enumerate(output):
            if torch.argmax(i) == y[idx]:
                correct += 1
            total += 1

print("Accuracy: ", round(correct/total, 2))
```

```
Accuracy:  0.96
```

Упражнение 5

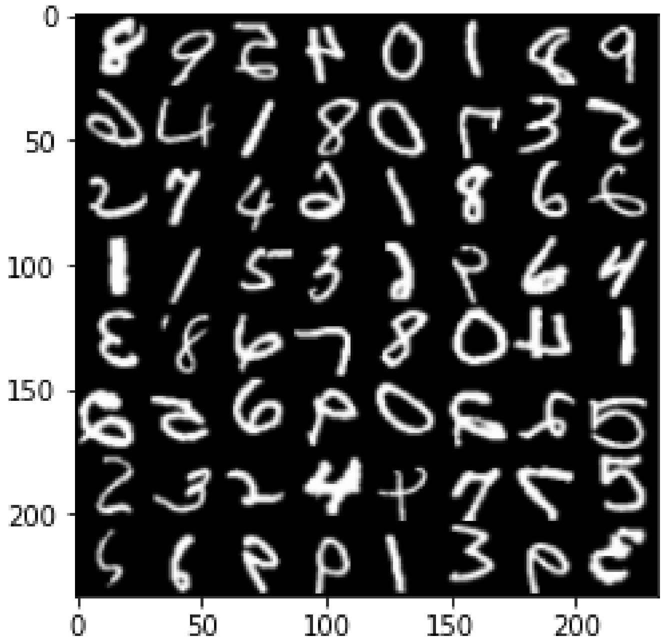
3. Попробуй преобразовать сеть в автокодировщик (автоэнкодер).

   для преобразования вектора в изображение используйте X.view (-1,28,28)

Помимо работы с набором данных как таковым, мы можем увеличить размер набора данных с помощью методов аументации. Эта операция может выполняться статически (перед обучением, не рекомендуется) или динамически (во время загрузки батчей). Последнее можно сделать с помощью преобразований `transforms`, как показано ниже.

```
transform = transforms.Compose([
                                transforms.Resize((32, 32)),
                                transforms.RandomCrop(28),
                                transforms.RandomHorizontalFlip(),
                                transforms.ToTensor(),
                               ])


train = torchvision.datasets.MNIST('',
                                   train=True,
                                   download=True,
                                   transform  = transform)

test  = torchvision.datasets.MNIST('',
                                   train=False,
                                   download=True,
                                   transform =transform)

trainset = torch.utils.data.DataLoader(train, batch_size=BATCH_SIZE,
shuffle=True)

testset  = torch.utils.data.DataLoader(test, batch_size=BATCH_SIZE,
shuffle=False)

images, _ = dataiter = next(iter(trainset))
image_show(torchvision.utils.make_grid(images, nrow = 8, padding = 1))
```

```
(233, 233, 3)
```



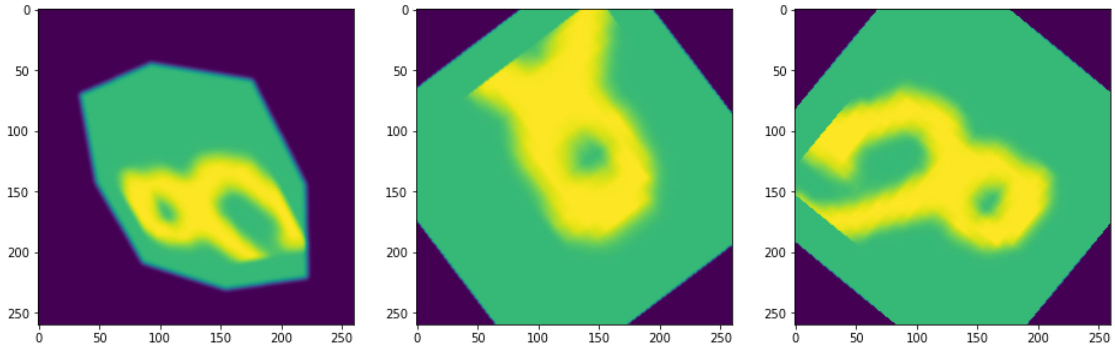
Ниже вы можете найти более сложный конвейер преобразований. Отметим, что *torch* позволяет нативно работать с форматами *numpy*, *pillow (PIL)* и тензорами изображений.

```
import torchvision.transforms as T

to_pil = T.ToPILImage()
img = to_pil(images[0])

padding = 30
kernel_size = 53
preprocess = T.Compose([
                T.ToTensor(),
                T.Resize(300),
                T.CenterCrop(250),
                T.RandomSizedCrop(200),
                T.Pad(padding=padding),
                lambda x:x**2+2,
                T.RandomRotation(degrees=180),
                T.RandomPerspective(distortion_scale=0.6, p=0.3),
                T.GaussianBlur(kernel_size=kernel_size),

T.RandomApply([T.RandomErasing(),T.RandomHorizontalFlip()],p=0.6),
                T.Normalize(mean=[0.5],std=[0.225])
                 ])

plt.figure(figsize = (16,8))

x = preprocess(img)
plt.subplot(131);plt.imshow(x[0,:,:])

x = preprocess(img)
```

```
plt.subplot(132); plt.imshow(x[0,:,:])

x = preprocess(img)
plt.subplot(133); plt.imshow(x[0,:,:])
```

```
C:\ProgramData\Anaconda3\lib\site-packages\torchvision\transforms\functional_tensor.py:876: UserWarning: Argument fill/fillcolor is not supported for Tensor input. Fill value is zero
  warnings.warn("Argument fill/fillcolor is not supported for Tensor input. Fill value is zero")
```

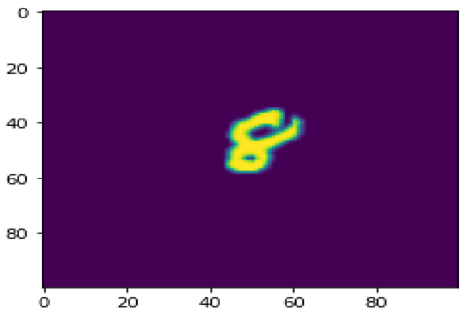
```
<matplotlib.image.AxesImage at 0x2486f48bac8>
```



Фактически, вы можете работать с изображениями как с объектом *PIL*, а затем преобразовать его обратно в тензор.

```
from PIL import Image

to_pil = transforms.ToPILImage()
img = to_pil(images[0])

transforms.ToTensor()

center_crops = transforms.CenterCrop(size=100)(img)
center_crops.show()

torch_image = transforms.ToTensor()(center_crops)
plt.imshow(torch_image[0,:,:])
```

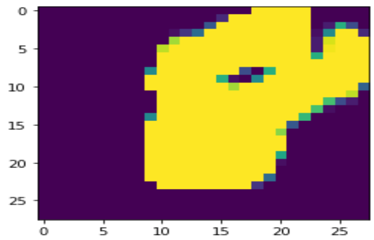
```
<matplotlib.image.AxesImage at 0x2486fa012c8>
```



также вы можете использовать преобразования как отдельные функции для тензорных изображений

```
new_img = T.functional.adjust_brightness(img,brightness_factor=10)
plt.imshow(new_img)
```

```
<matplotlib.image.AxesImage at 0x2486f784888>
```



Упражнение 6

4. Попытайтесь повысить точность нейронной сети (показанной выше), используя аугментацию для увеличения набора данных.

Лабораторная 4

Работа сопровождается документом в формате ipnb

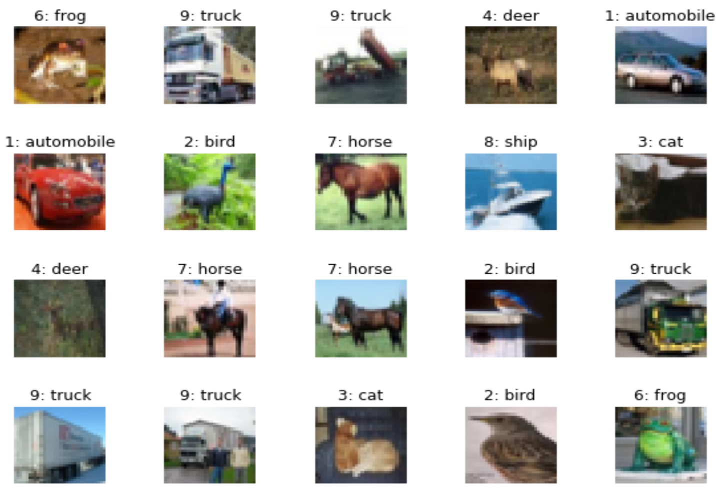
Сверточные нейронные сети в PyTorch

Сверточная нейронная сеть классификации изображений

```
import matplotlib.pyplot as plt
import numpy as np

import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim

import torchvision
import torchvision.transforms as transforms

try:
    from torchsummary import summary
except:
  !pip install -U torchsummary

    from torchsummary import summary
```

Рассмотрим задачу классификации для датасета Cifar10 - это классический датасет с изображениями, относящимися к 10 классам. Каждое изображение трех-цветное и имеет размер 32 на 32 пикслея (то есть представляет тензор размером 3x32x32).

Например Cifar10 содержит следующие изображения.



Загрузим датасет и проведем его нормализацию

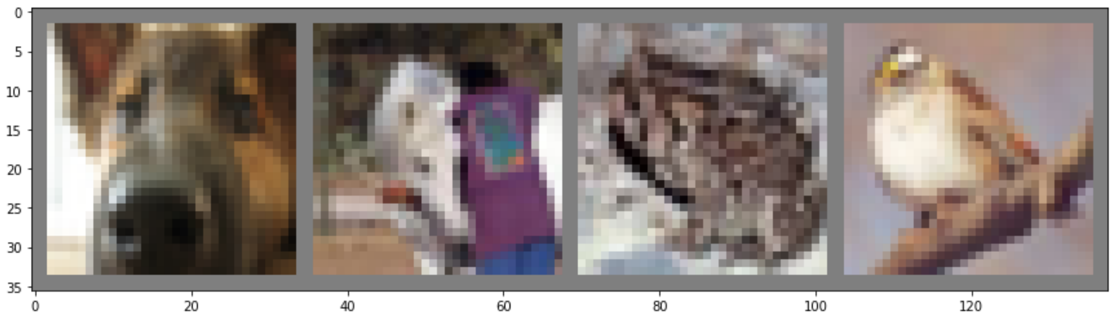
```
transform =\
    transforms.Compose([transforms.ToTensor(),
                        transforms.Normalize(
                            mean = (0.5, 0.5, 0.5),
                            std  = (0.5, 0.5, 0.5)  )
                       ])

BATCH_SIZE = 4
```

```
trainset = torchvision.datasets.CIFAR10(root      = './data',
                                        train     = True,
                                        download  = True,
                                        transform = transform)

trainloader = torch.utils.data.DataLoader(trainset,
                                          batch_size = BATCH_SIZE,
                                          shuffle    = True)

testset = torchvision.datasets.CIFAR10(root      = './data',
                                       train     = False,
                                       download  = True,
                                       transform = transform)

testloader = torch.utils.data.DataLoader(testset,
                                         batch_size = BATCH_SIZE,
                                         shuffle    = False)

classes = ('plane', 'car', 'bird', 'cat','deer', 'dog', 'frog',
'horse', 'ship', 'truck')
```

```
Files already downloaded and verified
Files already downloaded and verified
```

Посмотрим на примеры изображений

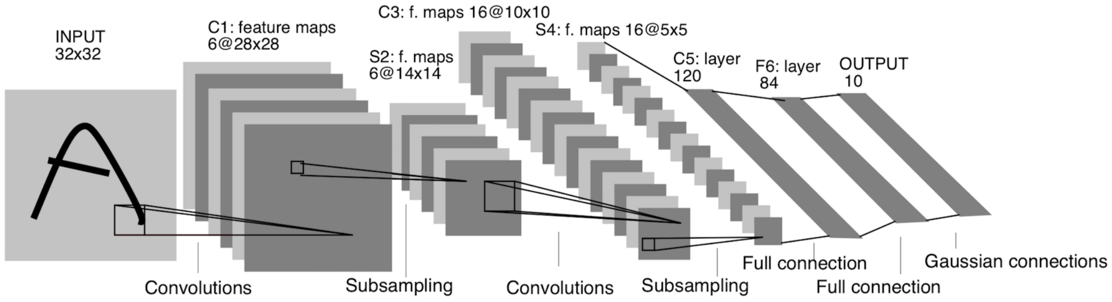
```
def imshow(img):
    img = img / 2 + 0.5     # unnormalize
    npimg = img.numpy()

    plt.figure(figsize=(16,4))
    plt.imshow(np.transpose(npimg, (1, 2, 0)))
    plt.show()

batch = trainloader

dataiter = iter(trainloader)
images, labels = dataiter.next()

imshow(torchvision.utils.make_grid(images))

print(' '.join('%5s' % classes[labels[j]] for j in range(4)))
```



```
dog horse  frog  bird
```

Теперь опишем класс сети. Будем использовать классическую сеть LeNet. Данная сеть была предложена в 1998 году и является одной из первых попыток разработки современного deep learning.

Сеть имеет следующую архитектуру



Таким образом сеть имеет:

- входной слой для изображений 32x32 (в нашем случае 3x32x32),
- два сверточных слоя,
- каждый сверточный слой имеет также макс-пулинг субдискретизацию,
- слой векторизации карты признаков (приобразования матрицы в вектор путем ее "разворачивания",
- два полносвязных внутренних слоя,
- выходной слой - слой классификации с 10 выходами.

Также особенности архитектуры:

- свертка валидная (с уменьшением размера карты признаков на размер ядра - 1)*;
- ядро свертки имеет размер 5x5 (размер ядра = 5);
- макс-пулин выполняется с шагом 2 по тайлам 2x2;
- первый сверточный слой выдает 6 карт признаков 14x14 (после макс-пулинга);
- второй сверточный слой выдает 16 карт признаков 5x5 (после макс-пулинга);
- слой разворачивания принимает 16 карт признаков 5x5 и выдет вектор 1x16 * 5 * 5 (1x400);
- первый полносвязный слой имеет 120 выходов (120 перцептронов);
- второй полносвязный слой имеет на выходе 84 перцептрона.

*Вобщем случае размер выхода свертки можно расчитать как: W=(W−F+2P)/S+1, where W is input size, F is kernel size, S is stride applied, and P is padding.

Реализуем сверточную сеть LeNet в виде класса, !Однако, для учебных целей добавим в сеть батч-нормализацию и дроп-ауты

```
class CNNModel(nn.Module):

    N_CLASSES = 10

    def __init__(self):
        super(CNNModel, self).__init__()
```

```python
        # convolution
        self.conv1   = nn.Conv2d(in_channels  = 3,
                                 out_channels = 6,
                                 kernel_size  = 5)

        self.conv2   = nn.Conv2d(in_channels  = 6,
                                 out_channels = 16,
                                 kernel_size  = 5)

        self.bn  = nn.BatchNorm2d(16)
        self.dropout = nn.Dropout(0.1)

        self.maxpool = nn.MaxPool2d(kernel_size = 2,
                                    stride      = 2)

        # classification
        self.fc1 = nn.Linear(in_features  = 16 * 5 * 5,
                             out_features = 120)

        self.fc2 = nn.Linear(in_features  = 120,
                             out_features = 84)

        self.fc_out = nn.Linear(in_features = 84,
                                out_features = self.N_CLASSES)

    # specific operation
    def flatten(self, x):
        return x.view(-1, 16 * 5 * 5)

    def forward(self, x):

        # 1-st layer
        x = self.conv1(x)
        x = torch.relu(x)
        x = self.maxpool(x)
        x = self.dropout(x)

        #2-nd layer
        x = self.conv2(x)
        x = self.bn(x)
        x = torch.relu(x)
        x = self.maxpool(x)

        #flatten
        x = self.flatten(x)

        # 1-st fc layer
        x = self.fc1(x)
        x = torch.relu(x)
```

```
        # 2-nd fc layer
        x = self.fc2(x)
        x = torch.relu(x)

        # output layer
        x = self.fc_out(x)

        return x#torch.softmax(x, dim=1)
```

```
print(images.shape)
```

```
torch.Size([4, 3, 32, 32])
```

Опишим настрой

```
LR = 0.001
cnn_net   = CNNModel()
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(cnn_net.parameters(), lr=LR)
```

Посмотрим структуру параметров созданной сети

```
for name, param in cnn_net.named_parameters():
    print(name, '\t structure =',list(param.size()))
```

```
conv1.weight      structure = [6, 3, 5, 5]
conv1.bias        structure = [6]
conv2.weight      structure = [16, 6, 5, 5]
conv2.bias        structure = [16]
bn.weight    structure = [16]
bn.bias      structure = [16]
fc1.weight        structure = [120, 400]
fc1.bias     structure = [120]
fc2.weight        structure = [84, 120]
fc2.bias     structure = [84]
fc_out.weight     structure = [10, 84]
fc_out.bias       structure = [10]
```

```
summary(cnn_net,input_size=(3, 32, 32))
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1            [-1, 6, 28, 28]             456
         MaxPool2d-2            [-1, 6, 14, 14]               0
           Dropout-3            [-1, 6, 14, 14]               0
            Conv2d-4           [-1, 16, 10, 10]           2,416
       BatchNorm2d-5           [-1, 16, 10, 10]              32
         MaxPool2d-6             [-1, 16, 5, 5]               0
            Linear-7                  [-1, 120]          48,120
            Linear-8                   [-1, 84]          10,164
            Linear-9                   [-1, 10]             850
================================================================
Total params: 62,038
```

```
Trainable params: 62,038
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 0.08
Params size (MB): 0.24
Estimated Total Size (MB): 0.33
----------------------------------------------------------------
```

Процесс обучения

```python
EPOHS = 5

for epoch in range(EPOHS):  # loop over the dataset multiple times
    cnn_net.train()
    running_loss = 0.0
    for i, data in enumerate(trainloader, 0):

        # get the inputs; data is a list of [inputs, labels]
        inputs, labels = data

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = cnn_net(inputs)
        loss    = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        # print statistics
        running_loss += loss.item()

        if i % 2000 == 1999:    # print every 2000 mini-batches
            print('[%d, %5d] loss: %.3f' %
                  (epoch + 1, i + 1, running_loss / 2000))
            running_loss = 0.0

print('Finished Training')
```

```
[1,  2000] loss: 1.894
[5, 12000] loss: 1.165
Finished Training
```

Сохраним результат

```python
PATH = './cifar_net.pth'
torch.save(cnn_net.state_dict(), PATH)
```

Загрузим результат

```python
cnn_net = CNNModel()
cnn_net.load_state_dict(torch.load(PATH))
```

```
<All keys matched successfully>
```

Проверим результат на тесте

```
cnn_net.eval()
dataiter = iter(testloader)
images, labels = dataiter.next()
outputs = cnn_net(images)
print(outputs)
```

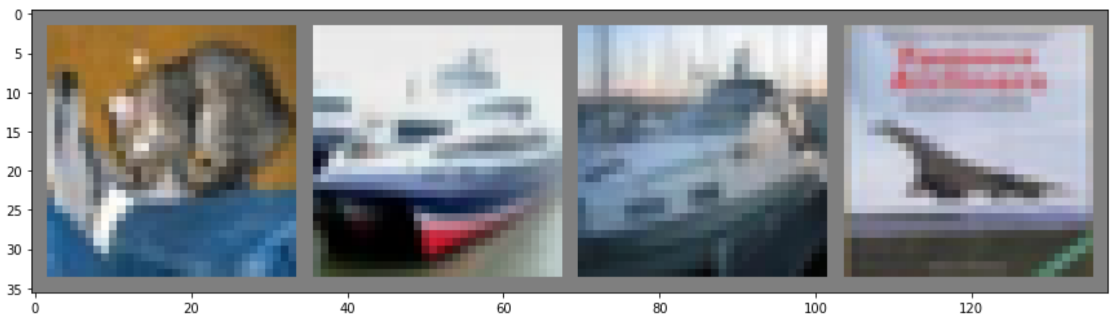
Приведем результат к более понятному виду Каждый номер в выходном тензоре будет класс соответствующий картинке батча

```
_, predicted = torch.max(outputs, 1)
print(predicted)
```

```
tensor([3, 1, 8, 8])
```

Посмотрим, что получилось

```
# print images
imshow(torchvision.utils.make_grid(images))
print('GroundTruth: ', ' '.join('%5s' % classes[labels[j]] for j in
range(4)))
print('Predicted :  ', ' '.join('%5s' % classes[predicted[j]] for j in
range(4)))
```



```
GroundTruth:    cat  ship  ship plane
Predicted :     cat   car  ship  ship
```

Попробуем посчитать точность как число правильно классифицированных изобаржений к общему числу изображений

```
cnn_net.eval()
correct = 0
total   = 0
with torch.no_grad():
    for data in testloader:
        images, labels = data
        outputs         = cnn_net(images)
        _, predicted     = torch.max(outputs.data, 1)
        total    += labels.size(0)
        correct += (predicted == labels).sum().item()
```

```
print('Accuracy of the network on the 10000 test images: %d %%' % (
    100 * correct / total))
```

```
Accuracy of the network on the 10000 test images: 59 %
```

Проведем анализ по каждому классу

```
cnn_net.eval()
class_correct = list(0. for i in range(10))
class_total   = list(0. for i in range(10))
with torch.no_grad():
    for data in testloader:
        images, labels = data
        outputs = cnn_net(images)
        _, predicted = torch.max(outputs, 1)
        c = (predicted == labels).squeeze()
        for i in range(4):
            label = labels[i]
            class_correct[label] += c[i].item()
            class_total[label]   += 1


for i in range(10):
    print('Accuracy of %5s : %2d %%' % (
        classes[i], 100 * class_correct[i] / class_total[i]))
```

```
Accuracy of plane : 76 %
Accuracy of   car : 83 %
Accuracy of  bird : 52 %
Accuracy of   cat : 30 %
Accuracy of  deer : 49 %
Accuracy of   dog : 43 %
Accuracy of  frog : 67 %
Accuracy of horse : 70 %
Accuracy of  ship : 65 %
Accuracy of truck : 57 %
```

Упражнение 1

Добавьте к процессу обучения валидацию и добавьте валидационный датасет.

Упражнение 2

Проанализируйте полученные результаты, уберите из обучающей выборки класс с наименьшей точностью, проведите повторное обучение.

Оптимизация обучения модели сверточной сети

Инициализация весов модели

```
cnn_net = CNNModel()

def init_weights(m):
```

```
    # for differernt types of layers different initialization could be
applied
    if type(m) == nn.Linear:
        torch.nn.init.kaiming_normal_(m.weight)
        m.bias.data.fill_(0.01)

    #same as type(m) == nn.Conv2d, but more correct to use isinstance
    if isinstance(m, nn.Conv2d):
        torch.nn.init.xavier_uniform_(m.weight)
        torch.nn.init.zeros_(m.bias)

cnn_net.apply(init_weights);

criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(cnn_net.parameters(), lr=0.001)
```

Проверим как прошла инициализация для значений смещений

```
for name, p in cnn_net.named_parameters():
    if 'bias' in name:
        print(p,'\n')
```

Также посмортим веса для одного из слоев

```
print(cnn_net.fc2.weight)
```

Регуляризаия

используем эластическую регуляризацию для весов и для смещения.

```
EPOHS = 5
lambda_l1 = 1e-5
lambda_l2 = 1e-3
for epoch in range(EPOHS):  # loop over the dataset multiple times
    cnn_net.train()
    running_loss = 0.0
    for i, data in enumerate(trainloader, 0):

        # get the inputs; data is a list of [inputs, labels]
        inputs, labels = data

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = cnn_net(inputs)
        loss    = criterion(outputs, labels)

        l1_regularization = 0
        l2_regularization = 0

        for p in cnn_net.parameters():
```

```
            l1_regularization += torch.abs(p).sum()
            l2_regularization += torch.pow(p,2).sum()

        loss += lambda_l1*l1_regularization +
lambda_l2*l2_regularization

        loss.backward()
        optimizer.step()

        # print statistics
        running_loss += loss.item()

        if i % 2000 == 1999:    # print every 2000 mini-batches
            print('[%d, %5d] loss: %.3f' %
                  (epoch + 1, i + 1, running_loss / 2000))
            running_loss = 0.0


print('Finished Training')
```

```
[1,  2000] loss: 2.244
[2, 10000] loss: 1.588

----------------------------------------------------------------------
-----
KeyboardInterrupt                         Traceback (most recent call
last)
<ipython-input-35-ace0efcf3cc4> in <module>
     21
     22         for p in cnn_net.parameters():
---> 23             l1_regularization += torch.abs(p).sum()
     24             l2_regularization += torch.pow(p,2).sum()
     25

KeyboardInterrupt:
```

посмотрим на результат

```
cnn_net.eval()
class_correct = list(0. for i in range(10))
class_total   = list(0. for i in range(10))
with torch.no_grad():
    for data in testloader:
        images, labels = data
        outputs = cnn_net(images)
        _, predicted = torch.max(outputs, 1)
        c = (predicted == labels).squeeze()
        for i in range(4):
            label = labels[i]
            class_correct[label] += c[i].item()
            class_total[label]   += 1

for i in range(10):
```

```
    print('Accuracy of %5s : %2d %%' % (
        classes[i], 100 * class_correct[i] / class_total[i]))
```

Упрпажнение 1

Проведите инициализацию смещения слоя батч-нормализации нулями.

Упрпажнение 2

5. Исключите из регуляризации слой батч-нормализации.
6. Введите особую процедуру регуляризации для выходного слоя сети Click here for hint Обратится к конкретному слою можно

```
torch.norm(model.fc1.weight, p=1)
```

Перенос обучения

На практике очень редко кто-то пытается обучить всю сверточную сеть с нуля (со случайной инициализацией), потому что относительно редко имеется набор данных достаточного размера и достаточное время/вычислительные рессурсы для обучения. Вместо этого обычно используют предварительно обученную сеть (предобученную на очень большом наборе данных, например, ImageNet, который содержит 1,2 миллиона изображений с 1000 категориями). Результаты предобученной сети используют либо как инициализацию, либо как фиксированный экстрактор признаков (кодировщик признаков).

```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.optim import lr_scheduler
import numpy as np
import torchvision
from torchvision import datasets, models, transforms
import matplotlib.pyplot as plt
from torchsummary import summary
```

Загрузим предварительно обученную модель ResNet18

```
resnet = torchvision.models.resnet18(pretrained=True)
summary(resnet,input_size=(3, 32, 32))

for name, param in resnet.named_parameters():
    print(name, '\t structure =',list(param.size()))
```

Теперь нам нужно заморозить все параметры для экстрактора признаков. Для этого нам нужно установить *requires_grad == False*, чтобы заморозить параметры, чтобы градиенты не вычислялись в *backward()*.

```
#freeze all parameters
for param in resnet.parameters():
    param.requires_grad = False
```

Для создания нашего собственного классификатора нам нужно сбросить последний полносвязанный слой.

```
# Parameters of newly constructed modules have requires_grad=True by
default
num_ftrs = resnet.fc.in_features

resnet.fc = nn.Linear(num_ftrs, len(classes))
```

```
---------------------------------------------------------------------
-----
NameError                                 Traceback (most recent call
last)
<ipython-input-7-64edd8e9d67f> in <module>
      2 num_ftrs = resnet.fc.in_features
      3
----> 4 resnet.fc = nn.Linear(num_ftrs, len(classes))

NameError: name 'classes' is not defined
```

Определим особенности дообучения сети. Будем обучать только последний слой.

```
resnet = resnet.to(device)

criterion = nn.CrossEntropyLoss()

# Observe that only parameters of final layer are being optimized as
# opposed to before.
optimizer_resnet = optim.SGD(model_conv.fc.parameters(), lr=0.001,
momentum=0.9)

# Decay LR by a factor of 0.1 every 7 epochs
exp_lr_scheduler = lr_scheduler.StepLR(optimizer_conv, step_size=7,
gamma=0.1)
```

```
---------------------------------------------------------------------
-----
NameError                                 Traceback (most recent call
last)
<ipython-input-9-e9dc103a3971> in <module>
----> 1 resnet = resnet.to(device)
      2
      3 criterion = nn.CrossEntropyLoss()
      4
      5 # Observe that only parameters of final layer are being
optimized as

NameError: name 'device' is not defined
```

```
EPOHS = 5

for epoch in range(EPOHS):  # loop over the dataset multiple times
    resnet.train()
```

```python
    running_loss = 0.0
    for i, data in enumerate(trainloader, 0):

        # get the inputs; data is a list of [inputs, labels]
        inputs, labels = data

        # zero the parameter gradients
        optimizer_resnet.zero_grad()

        # forward + backward + optimize
        outputs = resnet(inputs)

        loss    = criterion(outputs, labels)

        loss.backward()

        optimizer_resnet.step()

        # print statistics
        running_loss += loss.item()

        if i % 2000 == 1999:    # print every 2000 mini-batches
            print('[%d, %5d] loss: %.3f' %
                  (epoch + 1, i + 1, running_loss / 2000))
            running_loss = 0.0

print('Finished Training')

resnet.eval()
class_correct = list(0. for i in range(10))
class_total   = list(0. for i in range(10))

with torch.no_grad():
    for data in testloader:
        images, labels = data
        outputs = resnet(images)

        _, predicted = torch.max(outputs, 1)
        c = (predicted == labels).squeeze()

        for i in range(4):
            label = labels[i]
            class_correct[label] += c[i].item()
            class_total[label]   += 1


for i in range(10):
    print('Accuracy of %5s : %2d %%' % (
        classes[i], 100 * class_correct[i] / class_total[i]))
```

Упражнение 6

7. Переобучить сеть resnet полностью (используя предобученные параметры как инициализацию).
8. Сравните результаты с полученными выше.

Примечание Если у вас нет GPU, то рекомендуем использовать Google Colab с включенным GPU. Для смены среды выполнения (**runtime**) выбирите **Change runtime type** в соответствующем меню. После экспериментов не забудьте вернуть среду на *None* так как время выделяемое для работы GPU ограничено.

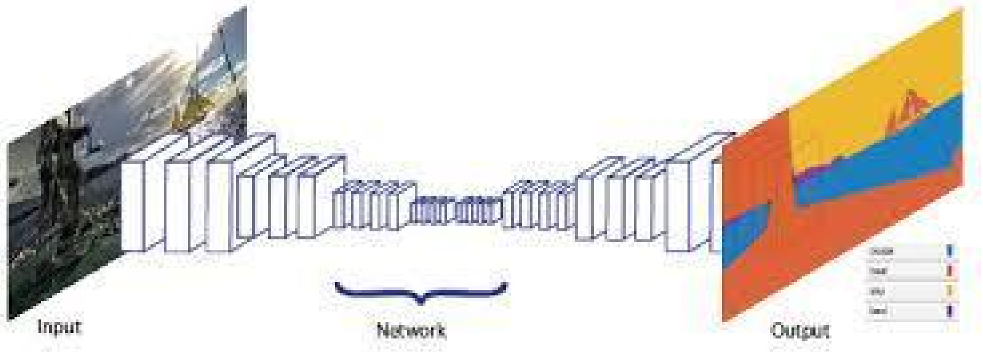
Лабораторная 5

Работа сопровождается документом в формате ipnb

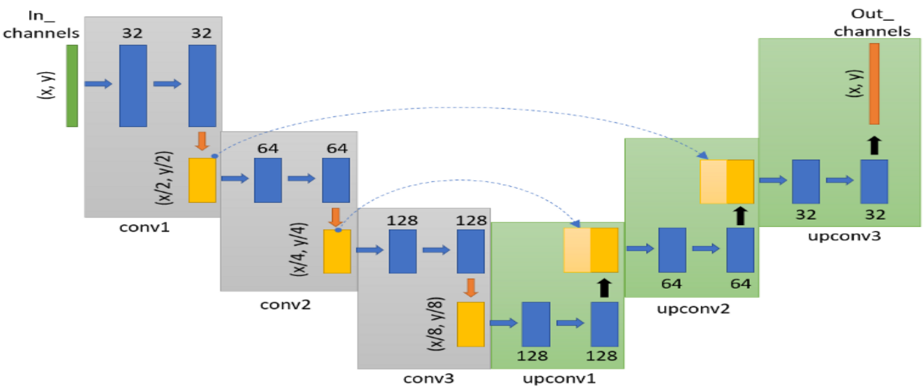
Сегментационные модели в PyTorch

Сегментационные модели в задачах компьютерного зрения. Изучение модели U-Net. Предобучение модели. Особенности переноса обучения для задач семантической сегментации. Изучение аугментации изображений в задачах семантической сегментации.

В предыдущих примерах работы с pytorch мы рассмотрели задачу классификации. Однако, это не едиственная задача, которую можно решать нейронными сетями. В данном примере рассмотрим задачу семантической сегментации.
В таких задачах требуется выделение каких-либо объектов в выходных данных (например, изображении) из входных данных (например, входного изображения).
Работу сети можно проиллюстрировать следующим образом.



Задачу сематической сегментации рассмотрим для модели нейронной сети UNet. Данная сеть, как и большинство сегментационных моделей, состоит из двух частей: энкодера и декодера.
Цель первой части (энкодера) - сжатие исходных данных в т.н. "набор карт латентных признаков".
То есть, по существу, закодировать входную информацию с избыточностью так, чтобы потом можно было раскодировать только то, что нужно - задано выходной маской.
Цель второй части (декдера) - раскодирование набора латентных признаков так, чтобы остался выделенным только нужный объект.
Особенность сети UNet заключается в том, что при раскодировании используется не только информацию декодера, но и информация из энкода. Это достигается за счет соединения слоев, получаемых в декодере непосредственно и аналогичных по размеру слоев энкодера.
Схема данной сети показана на рисунке ниже.



Импорт основных библиотек

```python
import torch
import torch.nn as nn
import torch.nn.functional as F
from torchsummary import summary
import torch.optim as optim
from torch.optim import lr_scheduler

from torch.utils.data import Dataset, DataLoader
from torchvision import transforms, datasets, models

import time
import copy
import numpy as np
import random
import matplotlib.pyplot as plt

from PIL import Image
```

Создадим датасет - будем использовать готовый пример, взятый тут, однако не много измененный.

Будем создавать изображения, на которых имеются разные геометрические фигуры. Допустим, что из этих фигур нас интересует треугольник. Поэтому мы будем генерировать изображения с разными фигурами в качестве входных и выходное изображение (т.н. маску) в виде треугольника.

```python
# Owned here https://github.com/usuyama/pytorch-unet

def generate_random_data(height, width, count):
    x, y = zip(*[generate_img_and_mask(height, width) for i in
range(0, count)])
    X = np.asarray(x) * 255
    X = X.transpose([0, 2, 3, 1]).astype(np.uint8)
    Y = np.asarray(y)
    return X, Y

def generate_img_and_mask(height, width):
    shape = (height, width)

    triangle_location = get_random_location(*shape, zoom = 2)
    circle_location1  = get_random_location(*shape, zoom=1)
    circle_location2  = get_random_location(*shape, zoom=0.7)
    mesh_location     = get_random_location(*shape, zoom=0.7)
    square_location   = get_random_location(*shape, zoom=0.4)
    plus_location     = get_random_location(*shape, zoom=1.2)

    # Create input image
    arr = np.zeros(shape, dtype=bool)
    arr = add_triangle(arr, *triangle_location)
    arr = add_circle(arr, *circle_location1)
    arr = add_circle(arr, *circle_location2, fill=True)
    arr = add_mesh_square(arr, *mesh_location)
    arr = add_filled_square(arr, *square_location)
```

```
    arr = add_plus(arr, *plus_location)
    arr = np.reshape(arr, (1, height, width)).astype(np.float32)

    # Create target masks
    masks = np.asarray([
        add_filled_square(np.zeros(shape, dtype=bool),
*square_location),
        add_circle(np.zeros(shape, dtype=bool), *circle_location2,
fill=True),
        add_triangle(np.zeros(shape, dtype=bool), *triangle_location),
        add_circle(np.zeros(shape, dtype=bool), *circle_location1),
        add_filled_square(np.zeros(shape, dtype=bool),
*mesh_location),
        add_plus(np.zeros(shape, dtype=bool),
*plus_location)]).astype(np.float32)

    return arr, masks

def add_square(arr, x, y, size):
    s = int(size / 2)
    arr[x-s,y-s:y+s] = True
    arr[x+s,y-s:y+s] = True
    arr[x-s:x+s,y-s] = True
    arr[x-s:x+s,y+s] = True
    return arr

def add_filled_square(arr, x, y, size):
    s = int(size / 2)
    xx, yy = np.mgrid[:arr.shape[0], :arr.shape[1]]
    return np.logical_or(arr, logical_and([xx > x - s, xx < x + s, yy
> y - s, yy < y + s]))

def logical_and(arrays):
    new_array = np.ones(arrays[0].shape, dtype=bool)
    for a in arrays:
        new_array = np.logical_and(new_array, a)
    return new_array

def add_mesh_square(arr, x, y, size):
    s = int(size / 2)
    xx, yy = np.mgrid[:arr.shape[0], :arr.shape[1]]
    return np.logical_or(arr, logical_and([xx > x - s, xx < x + s, xx
% 2 == 1, yy > y - s, yy < y + s, yy % 2 == 1]))

def add_triangle(arr, x, y, size):
    s = int(size / 2)
    triangle = np.tril(4*np.ones((size, size), dtype=bool))
    arr[x-s:x-s+triangle.shape[0],y-s:y-s+triangle.shape[1]] =
triangle
    return arr
```

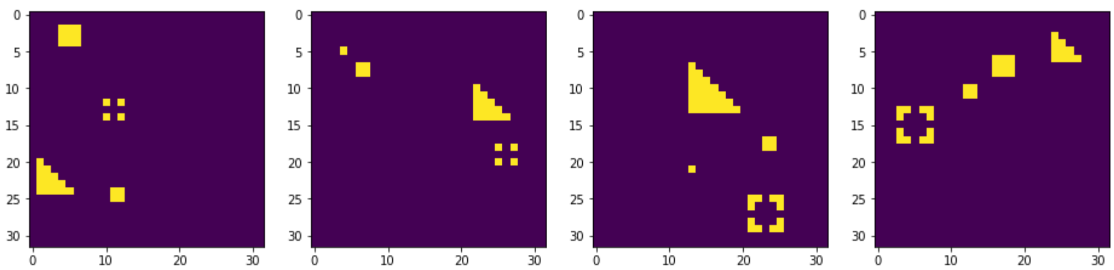
```
def add_circle(arr, x, y, size, fill=False):
    xx, yy = np.mgrid[:arr.shape[0], :arr.shape[1]]
    circle = np.sqrt((xx - x) ** 2 + (yy - y) ** 2)
    new_arr = np.logical_or(arr, np.logical_and(circle < size, circle
>= size * 0.7 if not fill else True))
    return new_arr

def add_plus(arr, x, y, size):
    s = int(size / 2)
    arr[x-1:x+1,y-s:y+s] = True
    arr[x-s:x+s,y-1:y+1] = True
    return arr

def get_random_location(width, height, zoom=1.0):
    x = int(width * random.uniform(0.1, 0.9))
    y = int(height * random.uniform(0.1, 0.9))
    size = int(min(width, height) * random.uniform(0.06, 0.12) * zoom)
    return (x, y, size)

class SimDataset(Dataset):
    def __init__(self, count, transform=None):
        self.input_images, self.target_masks =
generate_random_data(32, 32, count=count)
        self.transform = transform

    def __len__(self):
        return len(self.input_images)

    def __getitem__(self, idx):
        image = self.input_images[idx]
        mask  = self.target_masks[idx]

        if self.transform:
            image = self.transform(image)

        return [image, mask[:][2:3]]

# use the same transformations for train/val in this example
trans = transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize([0.5], [ 0.2])])

train_set = SimDataset(2000, transform = trans)
val_set  = SimDataset(200, transform = trans)

image_datasets = {'train': train_set, 'val': val_set}

batch_size = 8

dataloaders = {
    'train': DataLoader(train_set, batch_size=batch_size,
```

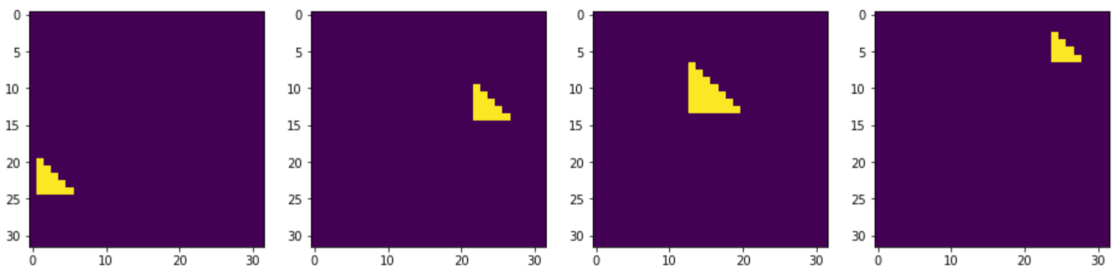
```
shuffle=True, num_workers=0),
    'val': DataLoader(val_set, batch_size=batch_size, shuffle=True,
num_workers=0)
}
```

Посмотрим на один минибатч из датасета - входные изображения и маски

```
images,masks = next(iter(dataloaders['train']))


print(images.shape)

print(images.shape)
print(masks.shape)

plt.figure(figsize=(16,4))
plt.subplot(141);plt.imshow(images[0][0])
plt.subplot(142);plt.imshow(images[1][0])
plt.subplot(143);plt.imshow(images[2][0])
plt.subplot(144);plt.imshow(images[3][0])

print(images.shape)
plt.figure(figsize=(16,4))
plt.subplot(141);plt.imshow(masks[0][0])
plt.subplot(142);plt.imshow(masks[1][0])
plt.subplot(143);plt.imshow(masks[2][0])
plt.subplot(144);plt.imshow(masks[3][0])
```

```
torch.Size([8, 1, 32, 32])
torch.Size([8, 1, 32, 32])
torch.Size([8, 1, 32, 32])
torch.Size([8, 1, 32, 32])

<matplotlib.image.AxesImage at 0x1d8ae2720c8>
```





Создадим класс нейронной сети

- Сеть будет состоять из 4 блоков энкодера и 4 блоков декотора.

- Каждый блок будет состоять из двух операций свертки с ядром 3x3 и режмом same.
- В энкодере будем использовать макспулин (MaxPool2d).
- В декодере операцию, обратную макспулингу - (Upsample) методом билинейного преобразования.
- Выберем "одинарный" размер (число) карт признаков как SIZE = 32.

```
def conv_block(in_channels, out_channels):
    return nn.Sequential(
        nn.Conv2d(in_channels,  out_channels, 3, padding=1),
        nn.ReLU(inplace=True),
        nn.Conv2d(out_channels, out_channels, 3, padding=1),
        nn.ReLU(inplace=True)
    )

SIZE = 32

class UNet(nn.Module):

    def __init__(self, n_class=1):
        super().__init__()

        self.down1 = conv_block(1,      SIZE)
        self.down2 = conv_block(SIZE,   SIZE*2)
        self.down3 = conv_block(SIZE*2, SIZE*4)
        self.down4 = conv_block(SIZE*4, SIZE*8)

        self.maxpool  = nn.MaxPool2d(2)
        self.upsample = nn.Upsample(scale_factor=2,
                                    mode='bilinear',
                                    align_corners=True)

        self.up3 = conv_block(SIZE*4 + SIZE*8, SIZE*4)
        self.up2 = conv_block(SIZE*2 + SIZE*4, SIZE*2)
        self.up1 = conv_block(SIZE*2 + SIZE,   SIZE)

        self.out = nn.Conv2d(SIZE, n_class, 1)


    def forward(self, x):

        #ENCODER
        conv1 = self.down1(x)
        x = self.maxpool(conv1)

        conv2 = self.down2(x)
        x = self.maxpool(conv2)

        conv3 = self.down3(x)
        x = self.maxpool(conv3)

        x = self.down4(x)
```

```
        #DECODER
        x = self.upsample(x)
        x = torch.cat([x, conv3], dim=1)

        x = self.up3(x)
        x = self.upsample(x)
        x = torch.cat([x, conv2], dim=1)

        x = self.up2(x)
        x = self.upsample(x)
        x = torch.cat([x, conv1], dim=1)

        x = self.up1(x)

        out = self.out(x)

        return out
```

Создадим экземпляр нейронной сети

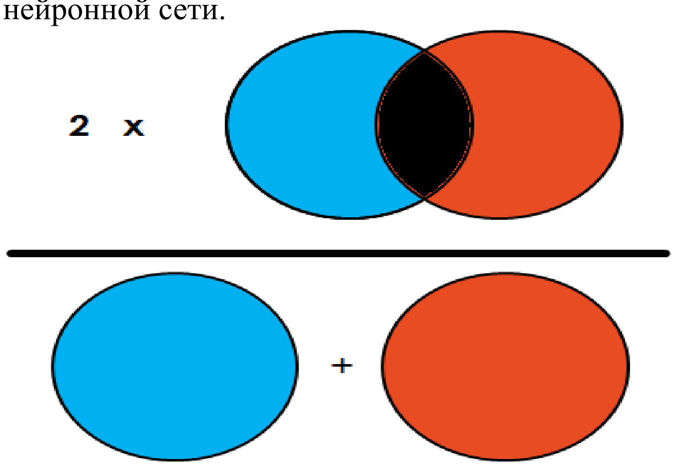
```
unet_model = UNet()
summary(unet_model, input_size=(1, 32, 32),device='cpu')
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 32, 32, 32]             320
              ReLU-2           [-1, 32, 32, 32]               0
            Conv2d-3           [-1, 32, 32, 32]           9,248
              ReLU-4           [-1, 32, 32, 32]               0
         MaxPool2d-5           [-1, 32, 16, 16]               0
            Conv2d-6           [-1, 64, 16, 16]          18,496
              ReLU-7           [-1, 64, 16, 16]               0
            Conv2d-8           [-1, 64, 16, 16]          36,928
              ReLU-9           [-1, 64, 16, 16]               0
        MaxPool2d-10             [-1, 64, 8, 8]               0
           Conv2d-11            [-1, 128, 8, 8]          73,856
             ReLU-12            [-1, 128, 8, 8]               0
           Conv2d-13            [-1, 128, 8, 8]         147,584
             ReLU-14            [-1, 128, 8, 8]               0
        MaxPool2d-15            [-1, 128, 4, 4]               0
           Conv2d-16            [-1, 256, 4, 4]         295,168
             ReLU-17            [-1, 256, 4, 4]               0
           Conv2d-18            [-1, 256, 4, 4]         590,080
             ReLU-19            [-1, 256, 4, 4]               0
         Upsample-20            [-1, 256, 8, 8]               0
           Conv2d-21            [-1, 128, 8, 8]         442,496
             ReLU-22            [-1, 128, 8, 8]               0
           Conv2d-23            [-1, 128, 8, 8]         147,584
             ReLU-24            [-1, 128, 8, 8]               0
         Upsample-25          [-1, 128, 16, 16]               0
```

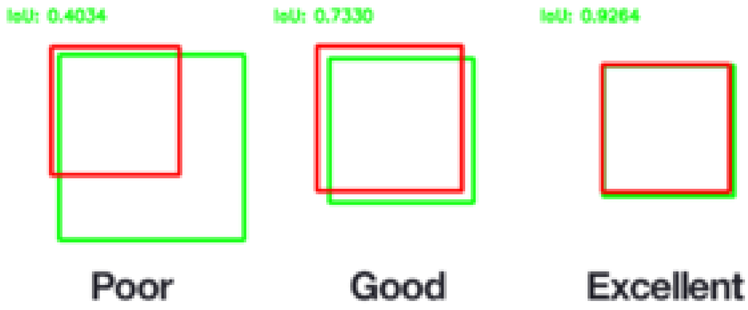
```
          Conv2d-26           [-1, 64, 16, 16]         110,656
            ReLU-27           [-1, 64, 16, 16]               0
          Conv2d-28           [-1, 64, 16, 16]          36,928
            ReLU-29           [-1, 64, 16, 16]               0
        Upsample-30           [-1, 64, 32, 32]               0
          Conv2d-31           [-1, 32, 32, 32]          27,680
            ReLU-32           [-1, 32, 32, 32]               0
          Conv2d-33           [-1, 32, 32, 32]           9,248
            ReLU-34           [-1, 32, 32, 32]               0
          Conv2d-35            [-1, 1, 32, 32]              33
================================================================
Total params: 1,946,305
Trainable params: 1,946,305
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 4.62
Params size (MB): 7.42
Estimated Total Size (MB): 12.05
----------------------------------------------------------------
```

Обучение будем проводить по двум вариантам функции потерь: бинарная энтропия и т.н. DICE

Dice (Коэффициент Серенса-Дайса) представляет собой метрику, характерную для сегментации - по существу, отношение удвоенной площади пересечения искомого объекта (макси) к сумме площадей искомого объекта и полученного по результатам работы нейронной сети.



Если говорить более качественно, то Dice говорит о том, на сколько правильно объект выделен по площади и по его расположению относительно искомого. Иллюстрация плохого, среднего и хорошего DICE приведена на картинке ниже



Расстояние, соответствующиее Dice расчитчвается как:

$$dice = 1 - \frac{2 \cdot (A \cap B)}{|A| + |B|}$$

где A, B - это площади полученного и искомого объектов, знак $\cap$ - это обозначение пересечения площадей

```
def dice_loss(pred, target, smooth = 1.):
    pred   = pred.contiguous()
    target = target.contiguous()

    intersection   = (pred * target).sum(dim=2).sum(dim=2)
    sum_of_squares = pred.sum(dim=2).sum(dim=2) +
target.sum(dim=2).sum(dim=2)

    loss = 1 - 2. * intersection / sum_of_squares

    return loss.mean()
```

Таким образом у нас будет две метрики DICE и BCE, а также функция потерь. Будем считать функцию потерь как комбинацию DICE и BCE с рзными весами.

Таким образом в функции потерь будет учитываться место объекта и его площадь (DICE), а также особенности формы объекта с точки зрения его параметров, например яркости или особенностей формы (например широховатости краев) (BCE).

```
METRICS_DEFAULT = {'bce':0.0,'dice':0.0,'loss':0.0}
```

Объеденим расчет метрик и потерь в одну функцию

```
def calc_loss(pred, target, metrics = None, bce_weight=0.5):

    if (metrics is None):
        metrics = METRICS_DEFAULT.copy()

    bce = F.binary_cross_entropy_with_logits(pred, target)

    pred = torch.sigmoid(pred)
    dice = dice_loss(pred, target)

    loss = bce * bce_weight + dice * (1 - bce_weight)

    metrics['bce']  += bce.data.cpu().numpy()
    metrics['dice'] += dice.data.cpu().numpy()
    metrics['loss'] += loss.data.cpu().numpy()
```

```
    return loss, metrics
```

Опишем процедуру тренировки как отдельную функцию

Процедура включает следующие параметры:

- модель нейронной сети (unet_model);
- словарь двух загрузчиков данных в формате DataLoader (dataloaders) (для валидации 'valid'; для тренировки 'train');
- функцию оптимизатор (optimizer);
- функцуию изменения сокрости обучения (scheduler);
- число эпох (num_epochs).

Выход процедуры - обученная модель.

Особенности процедуры train_model:

- Процедура включает как фазу тренировку, так и фазу валидацию для каждой эпохи.
- Каждый раз, когда после обучения эпохи значения метрик ниже чем были до этого, то результат тренировки записывается.
- В итоге возвращается результат на лучшей эпохе.
- В процессе каждой эпохи на экран выводится служебная информация.

```
def train_model(unet_model, dataloaders, optimizer,  scheduler=None, num_epochs=25):

    #INITIALIZATION OF THE BEST MODEL INITAL STATE
    best_model = copy.deepcopy(unet_model.state_dict())
    best_loss  = 1e10
    best_epoch = 0


    start_time = time.time()

    #EPOHS OF TRAINING
    for epoch in range(num_epochs):

        #SERVICE INFORMATION
        print('-' * 10)
        since = time.time()
        print('Epoch {}/{}'.format(epoch+1, num_epochs),end=' ')
        for param_group in optimizer.param_groups: print("LR", param_group['lr'])

        #TRAINING AND VALIDATION PHASES
        for phase in ['train', 'val']:

            #FOR SUCH LAYERS AS DROPOUT AND BATCH NORM
            if phase == 'train':
                unet_model.train()  # Set model to training mode
            else:
                unet_model.eval()   # Set model to evaluate mode
```

```
            #COUNTER FOR SAMPLES PROCESSED IN THE EPOCH
            epoch_samples = 0

            #RESET METRICS VALUES FOR EPOCH
            metrics       = METRICS_DEFAULT.copy()

            #MINI_BATCHES
            for inputs, labels in dataloaders[phase]:
                optimizer.zero_grad()

                #IF VAL, THEN NO_GRAD, ELSE GRAD_ENABLE
                with torch.set_grad_enabled(phase == 'train'):
                    outputs       = unet_model.forward(inputs)
                    loss, metrics = calc_loss(outputs, labels,
metrics)

                    if phase == 'train':
                        loss.backward()
                        optimizer.step()

                #FOR LOSS NORMALIZATION
                epoch_samples += inputs.size(0)

            #CHECK FOR THE BEST RESULT
            epoch_loss = metrics['loss'] / epoch_samples
            if phase == 'val' and epoch_loss < best_loss:
                print("saving best model")
                best_loss  = epoch_loss
                best_epoch = epoch
                best_model = copy.deepcopy(unet_model.state_dict())

            #LEARNING RATE SCHADULER
            if phase == 'train' and scheduler is not None:
                scheduler.step()

            #SERVICE INFORMATION
            print(str(phase),":",
                  'loss:%.5f'%(metrics['loss']),
                  'dice:%.5f'%(1-metrics['dice']), #1-DICE IS BETTER
TO INTERPRITATE
                  'bce:%.5f' %(metrics['bce']))

        #SERVICE INFORMATION
        time_elapsed = time.time() - since
        print('Epoch time: {:.0f}m {:.0f}s'.\
              format(time_elapsed // 60, time_elapsed % 60))

    #SERVICE INFORMATION
    print('Best val loss: {:4f}'.format(best_loss),
          'best epoch: %d'%(best_epoch),
```

```
            'training_time:{:.0f}m {:.0f}s'.\
                format((time.time()-start_time) // 60,
                       (time.time()-start_time) % 60))

    #LOAD THE BEST MODEL STATE (WEIGHTS AND BIASES)
    unet_model.load_state_dict(best_model)
    return unet_model
```

Зададим особенности обучения.

- Оптимизатор Adam;
- каждые 10 эпох будем снижать скорость обучения с коэффициентом 0.9;
- Сделаем возможность выбирать или использовать предобученная модель или инциализировать новую. То есть сделаем возможность дообучать модели.

```
EPOCH = 20
pretrained = True

# Observe that all parameters are being optimized
optimizer_ft = optim.Adam(unet_model.parameters(), lr=1e-4)

exp_lr_scheduler = lr_scheduler.StepLR(optimizer_ft, step_size=10,
gamma=0.9)

if pretrained == False: #IF NEW MODEL
    unet_model = UNet()
```

тренировка сети

```
unet_model =\
    train_model(unet_model,
                dataloaders = dataloaders,
                optimizer   = optimizer_ft,
                scheduler   = exp_lr_scheduler,
                num_epochs  = EPOCH)

----------
Epoch 1/20 LR 0.0001
train : loss:88.99715 dice:-149.26764 bce:27.72667
saving best model
val : loss:3.41808 dice:-5.21179 bce:0.62436
Epoch time: 0m 52s
----------
Epoch 20/20 LR 9e-05
train : loss:0.74586 dice:-0.37008 bce:0.12164
saving best model
val : loss:0.05855 dice:0.89247 bce:0.00957
Epoch time: 1m 53s
Best val loss: 0.000293 best epoch: 19 training_time:21m 44s
```

Создадим тестовый дата-сет

```python
test_dataset = SimDataset(4, transform = trans)
test_loader  = DataLoader(test_dataset, batch_size=4, shuffle=False,
num_workers=0)
```

Посмотрим на метрики на тестовом датасете

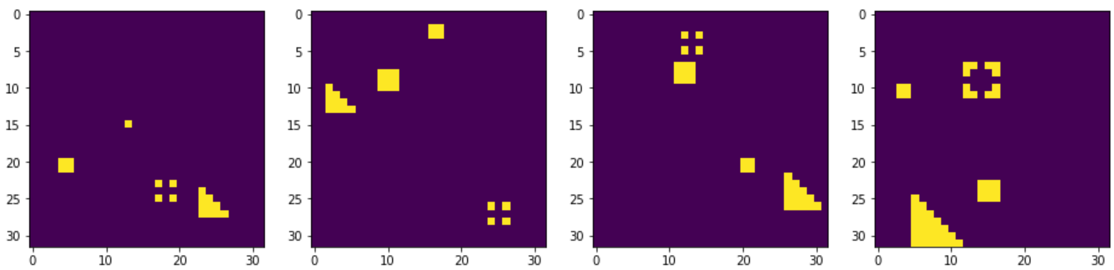
```python
unet_model.eval()
inputs, labels = next(iter(test_loader))
predicts       = unet_model(inputs)
loss, metrics  = calc_loss(predicts, labels)
print('loss:%.5f'%(metrics['loss']),
                  'dice:%.5f'%(1-metrics['dice']),
                  'bce:%.5f' %(metrics['bce']))
```

```
loss:0.00043 dice:0.99918 bce:0.00005
```

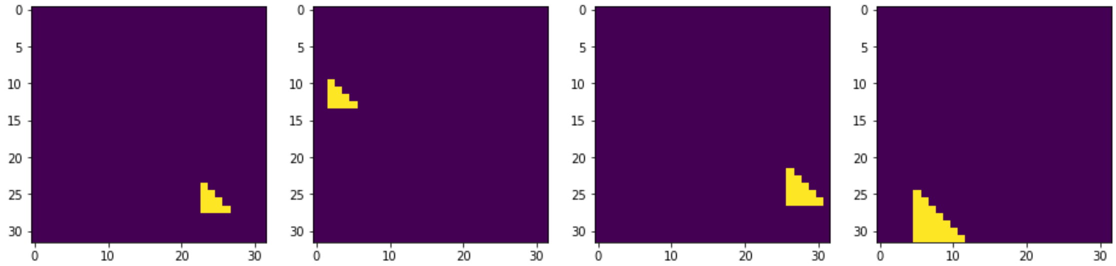
Проверим результат визуально

```python
predicts = predicts.data.cpu().numpy()

print(' I N P U T ')
plt.figure(figsize=(16,4))
plt.subplot(141);plt.imshow(inputs[0][0])
plt.subplot(142);plt.imshow(inputs[1][0])
plt.subplot(143);plt.imshow(inputs[2][0])
plt.subplot(144);plt.imshow(inputs[3][0])
plt.show()

print(' G R O U N D T R U T H ')
plt.figure(figsize=(16,4))
plt.subplot(141);plt.imshow(labels[0].sum(0))
plt.subplot(142);plt.imshow(labels[1].sum(0))
plt.subplot(143);plt.imshow(labels[2].sum(0))
plt.subplot(144);plt.imshow(labels[3].sum(0))
plt.show()

print(' P R E D I C T E D ')
plt.figure(figsize=(16,4))
plt.subplot(141);plt.imshow(predicts[0][0])
plt.subplot(142);plt.imshow(predicts[1][0])
plt.subplot(143);plt.imshow(predicts[2][0])
plt.subplot(144);plt.imshow(predicts[3][0])
plt.show()
```

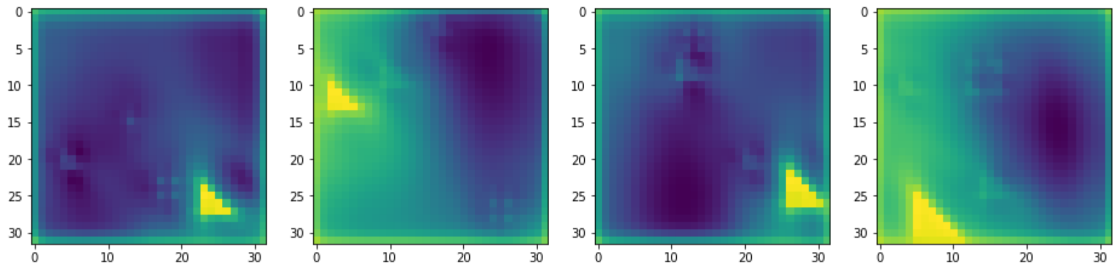
```
 I N P U T 
```



G R O U N D T R U T H



P R E D I C T E D



Упражнение 1

Сохраните веса модели в отдельный файл.

Дообучите модель (без ее сброса - перезагрузки) на 10 эпохах

Упражнение 2

Добавьте возможность вывода графика функции потерь и метрики DICE по результатам обучения.

Упражнение 3

Добавьте к процессу обучения функцию ранней остановки (остановки по неизменению или росту валидации)

Упражнение 4

Уберите из архитектуры UNet связи энкодера и декодера - то есть преобразуйте UNet в SegNet. Сравните качество обучения сети.

Предобучение сверточной сети автоэнкодером

Часто при обучении модели сети - для поднятия качества ее обучения, особенно для небольших датасетов желательным является предобучение нейронной сети. Такое предобучение может быть произведено или на каком-то близком, но большом датасете или, например, методом автоэнкодера (такой подход? по существу, представляет собой концепцию сетей глубокого доверия - deep belief network). В случае UNet подход глубокого доверия может быть произведен путем обучения нейронной сети для случая, когда ее выход должен повторять ее вход - по сути это обучение без учителя. Стоит отметить, что преобучение и обучение могут быть проведены с разными видами функции потерь или другими параметрами обучения.

Этап автоэнкодера

Выбирем в качестве функции потерь MSE (Minimum Square Error)

```
AUTOMETRICS_DEFAULT = {'mse':0.0,'dice':0.0,'loss':0.0}
```

Пересоздадим модель

```
unet_model = UNet()
```

Пусть функция потерь будет только MSE

```
def calc_autoloss(pred, inputs, metrics = None, mce_weight=1):

    if (metrics is None):
        metrics = AUTOMETRICS_DEFAULT.copy()

    mse = F.mse_loss(pred, inputs)

    pred = torch.sigmoid(pred)
    dice = dice_loss(pred, inputs)

    loss = mse #* mce_weight + dice * (1 - mce_weight)

    metrics['mse']  += mse.data.cpu().numpy()
    metrics['dice'] += dice.data.cpu().numpy()
    metrics['loss'] += loss.data.cpu().numpy()

    return loss, metrics
```

Перепишем процедуру обучения сети

```
def train_autoencoder(unet_model, dataloaders, optimizer,
scheduler=None, num_epochs=25):

    #INITIALIZATION OF THE BEST MODEL INITAL STATE
    best_model = copy.deepcopy(unet_model.state_dict())
    best_loss  = 1e10
    best_epoch = 0

    start_time = time.time()


    #EPOHS OF TRAINING
    for epoch in range(num_epochs):

        #SERVICE INFORMATION
        print('-' * 10)
        since = time.time()
        print('Epoch {}/{}'.format(epoch+1, num_epochs),end=' ')
        for param_group in optimizer.param_groups: print("LR",
```

```
param_group['lr'])

        #TRAINING AND VALIDATION PHASES
        for phase in ['train', 'val']:

            #FOR SUCH LAYERS AS DROPOUT  AND BATCH NORM
            if phase == 'train':
                unet_model.train()
            else:
                unet_model.eval()

            #COUNTER FOR SAMPLES PROCESSED IN THE EPOCH
            epoch_samples = 0

            #RESET METRICS VALUES FOR EPOCH
            metrics       = AUTOMETRICS_DEFAULT.copy()

            #MINI_BATCHES
            for inputs, labels in dataloaders[phase]:
                optimizer.zero_grad()

                #IF VAL, THEN NO_GRAD, ELSE GRAD_ENABLE
                with torch.set_grad_enabled(phase == 'train'):
                    outputs       = unet_model.forward(inputs)
                    loss, metrics = calc_autoloss(outputs, inputs,
metrics)

                    if phase == 'train':
                        loss.backward()
                        optimizer.step()

                #FOR LOSS NORMALIZATION ON THE BATCH SIZE
                epoch_samples += inputs.size(0)

            #CHECK FOR THE BEST RESULT
            epoch_loss = metrics['loss'] / epoch_samples
            if phase == 'val' and epoch_loss < best_loss:
                print("saving best model")
                best_loss  = epoch_loss
                best_epoch = epoch
                best_model = copy.deepcopy(unet_model.state_dict())

            #LEARNING RATE SCHADULER
            if phase == 'train' and scheduler is not None:
                scheduler.step()

            #SERVICE INFORMATION
            print(str(phase),":",
                  'loss:%.5f'%(metrics['loss']),
                  'dice:%.5f'%(1-metrics['dice']), #1-DICE IS BETTER
TO INTERPRITATE
```

```
                    'mse:%.5f' %(metrics['mse']))

        #SERVICE INFORMATION
        time_elapsed = time.time() - since
        print('Epoch time: {:.0f}m {:.0f}s'.\
              format(time_elapsed // 60, time_elapsed % 60))

    #SERVICE INFORMATION
    print('Best val loss: {:4f}'.format(best_loss),
          'best epoch: %d'%(best_epoch),
          'training_time:{:.0f}m {:.0f}s'.\
              format((time.time()-start_time) // 60,
                     (time.time()-start_time) % 60))

    #LOAD THE BEST MODEL STATE (WEIGHTS AND BIASES)
    unet_model.load_state_dict(best_model)
    return unet_model
```

Параметры обучения

```
EPOCH = 20
pretrained = True

# Observe that all parameters are being optimized
optimizer_ft     = optim.Adam(unet_model.parameters(), lr=1e-4)

exp_lr_scheduler = lr_scheduler.StepLR(optimizer_ft, step_size=10,
gamma=0.9)

if pretrained == False:
    unet_model = UNet()
```

Тренировка автоэнкодера

```
unet_model =\
    train_autoencoder(unet_model,
                      dataloaders = dataloaders,
                      optimizer   = optimizer_ft,
                      scheduler   = exp_lr_scheduler,
                      num_epochs  = EPOCH)

----------
Epoch 1/20 LR 0.0001
train : loss:6883.09349 dice:-222.18551 mse:6883.09349
saving best model
val : loss:3.59113 dice:-21.36611 mse:3.59113
Epoch time: 0m 51s
----------
Epoch 2/20 LR 0.0001
train : loss:21.05984 dice:-223.37881 mse:21.05984
saving best model
val : loss:1.06789 dice:-21.71218 mse:1.06789
Epoch time: 0m 52s
```

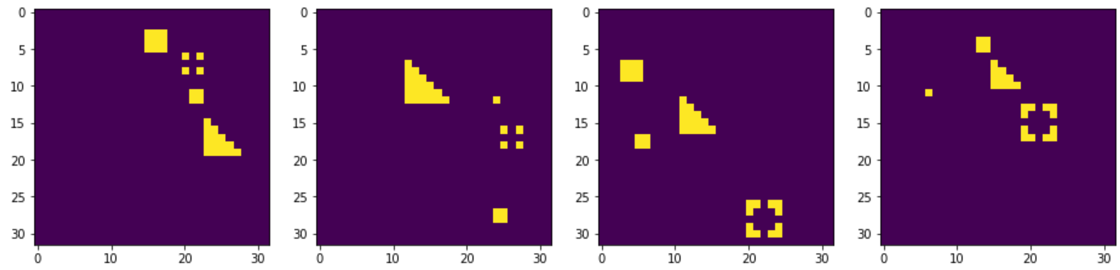
```
----------
Epoch 19/20 LR 9e-05
train : loss:0.73902 dice:-225.55331 mse:0.73902
saving best model
val : loss:0.06993 dice:-21.64289 mse:0.06993
Epoch time: 0m 49s
----------
Epoch 20/20 LR 9e-05
train : loss:0.67419 dice:-225.55898 mse:0.67419
saving best model
val : loss:0.06309 dice:-21.66396 mse:0.06309
Epoch time: 0m 50s
Best val loss: 0.000315 best epoch: 19 training_time:17m 52s
```

Посмотрим результат для тестового датасета, созданного ранее

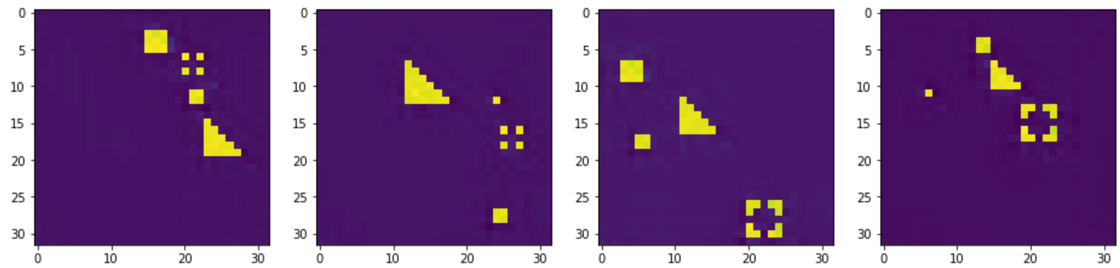
* Если тестовый датасет не найден, пресоздайте его

```
unet_model.eval()
inputs, labels = next(iter(test_loader))
predicts       = unet_model(inputs)
loss, metrics  = calc_autoloss(predicts, inputs)
print('loss:%.5f'%(metrics['loss']),
                  'dice:%.5f'%(1-metrics['dice']),
                  'mse:%.5f' %(metrics['mse']))

predicts = predicts.data.cpu().numpy()

print(' I N P U T  =   G R O U N D T R U T H ')
plt.figure(figsize=(16,4))
plt.subplot(141);plt.imshow(inputs[0][0])
plt.subplot(142);plt.imshow(inputs[1][0])
plt.subplot(143);plt.imshow(inputs[2][0])
plt.subplot(144);plt.imshow(inputs[3][0])
plt.show()

print(' P R E D I C T E D ')
plt.figure(figsize=(16,4))
plt.subplot(141);plt.imshow(predicts[0][0])
plt.subplot(142);plt.imshow(predicts[1][0])
plt.subplot(143);plt.imshow(predicts[2][0])
plt.subplot(144);plt.imshow(predicts[3][0])
plt.show()
```

```
loss:0.00181 dice:0.09802 mse:0.00181
 I N P U T  = G R O U N D T R U T H
```



P R E D I C T E D

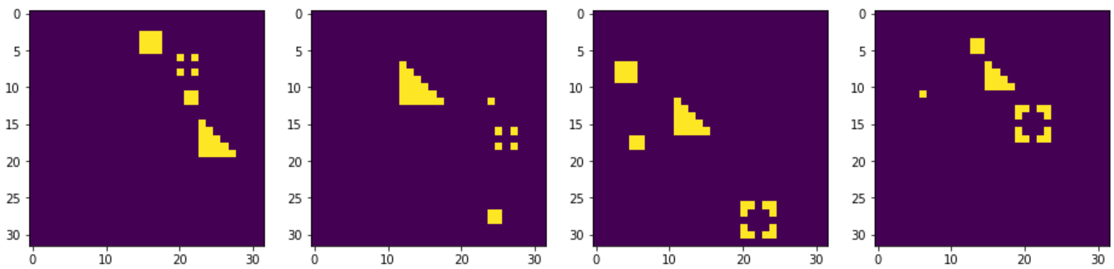


Попробуем обучить предобученную сеть

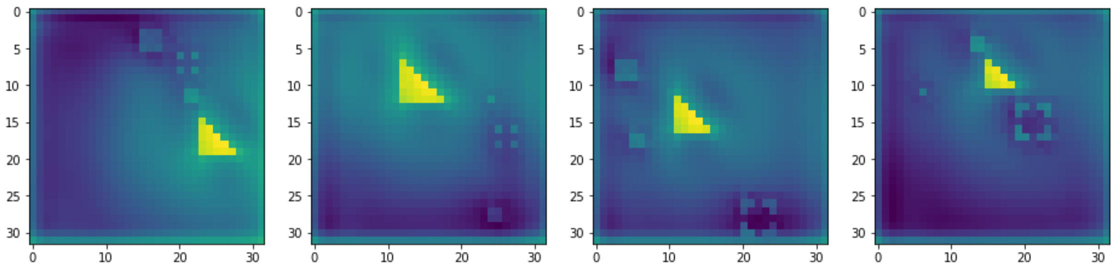
```
unet_model =\
    train_model(unet_model,
                dataloaders = dataloaders,
                optimizer   = optimizer_ft,
                scheduler   = exp_lr_scheduler,
                num_epochs  = EPOCH)

----------
Epoch 0/19 LR 8.1e-05
train : loss:9.51207 dice:-16.98738 bce:1.03676
saving best model
val : loss:0.38224 dice:0.26683 bce:0.03131
Epoch time: 0m 50s
----------
Epoch 1/19 LR 8.1e-05
train : loss:2.64964 dice:-4.07184 bce:0.22745
saving best model
val : loss:0.32173 dice:0.38086 bce:0.02432
Epoch time: 0m 55s
----------
Epoch 18/19 LR 7.290000000000001e-05
train : loss:0.42796 dice:0.19255 bce:0.04848
saving best model
val : loss:0.05664 dice:0.89400 bce:0.00727
Epoch time: 0m 52s
----------
Epoch 19/19 LR 7.290000000000001e-05
train : loss:0.45972 dice:0.13232 bce:0.05176
saving best model
val : loss:0.05570 dice:0.89581 bce:0.00720
Epoch time: 0m 51s
Best val loss: 0.000278 best epoch: 19 training_time:17m 31s
```

```
unet_model.eval()
inputs, labels = next(iter(test_loader))
predicts       = unet_model(inputs)
loss, metrics  = calc_loss(predicts, labels)
print('loss:%.5f'%(metrics['loss']),
                  'dice:%.5f'%(1-metrics['dice']),
                  'bce:%.5f' %(metrics['bce']))

predicts = predicts.data.cpu().numpy()

print(' I N P U T  = G R O U N D T R U T H ')
plt.figure(figsize=(16,4))
plt.subplot(141);plt.imshow(inputs[0][0])
plt.subplot(142);plt.imshow(inputs[1][0])
plt.subplot(143);plt.imshow(inputs[2][0])
plt.subplot(144);plt.imshow(inputs[3][0])
plt.show()

print(' P R E D I C T E D ')
plt.figure(figsize=(16,4))
plt.subplot(141);plt.imshow(predicts[0][0])
plt.subplot(142);plt.imshow(predicts[1][0])
plt.subplot(143);plt.imshow(predicts[2][0])
plt.subplot(144);plt.imshow(predicts[3][0])
plt.show()
```

```
loss:0.00000 dice:0.99999 bce:0.00000
 I N P U T  = G R O U N D T R U T H
```



```
 P R E D I C T E D
```



Упражнение 5

Сравните результат работы предобученной сети и сети с произвольной инициализацией параметров. Для этого загрузите и тот и другой в разные модели и проведите оценку функции потерь, метрик качества, а также проведите визуальное сравнение.

Упражнение 6

Добавьте к процедуре обучения нейронной сети аутментацию данных с использованием функций `transform`. Переобучите сеть и сравните результаты.

Лабораторная 6

Работа сопровождается документом в формате ipnb

Изучение особенностей задач поиска и локализации объектов на изображениях на основе библиотеки Detectron2

Задачи поиска и локализации объектов на изображениях. Особенности работы библиотеки Detectron2. Набор данных COCO. Изучение нейронных сетей Faster-RCNN (object detection), Mask-RCNN (instance segmentation) и FPN (Panoptic Segmentation).

Библиотека Detectron2 является high-api библиотекой на основе `PyTorch` для решения таких задач, как object detection (поиска и локализации объектов на изображениях), instance segmentation (экземплярно), keypoint segmentation и panoptic segmentationфото- и видео данных.
Библиотека разработана компанией facebook и имеет открытый исходный код. В основе библиотеки лжат фреймворк pytorch и caffe2 (на котором написана первая версия detectron).
Ссылки

- репозиторий библиотеки https://github.com/facebookresearch/Detectron2
- документация и сравнения с аналогичными фреймворками https://detectron2.readthedocs.io/notes/benchmarks.html
- репозиторий https://github.com/facebookresearch/detectron2/tree/master/projects
- https://ai.facebook.com/blog/-detectron2-a-pytorch-based-modular-object-detection-library-/ - информация на официальном сайте.
- предыдущая версия https://github.com/facebookresearch/Detectron/
- https://github.com/facebookresearch/maskrcnn-benchmark/ также фреймворк для сегментации, лежащий в основе.
- Установка Detectron2 на ваш ПК https://detectron2.readthedocs.io/tutorials/install.html

Пояснения к тому, что такое семантическая, экземплярная и паноптическая сегментации.

Разные виды сегментации/детекции, доступные в Detectron2

структура detectron2

```
├─checkpoint  <- checkpointer and model catalog handlers
├─config      <- default configs and handlers
├─data        <- dataset handlers and data loaders
├─engine      <- predictor and trainer engines
├─evaluation  <- evaluator for each dataset
```

```
├─export      <- converter of detectron2 models to caffe2 (ONNX)
├─layers      <- custom layers e.g. deformable conv.
├─model_zoo   <- pre-trained model links and handler
├─modeling
│  ├─meta_arch <- meta architecture e.g. R-CNN, RetinaNet
│  ├─backbone  <- backbone network e.g. ResNet, FPN
│  ├─proposal_generator <- region proposal network
│  └─roi_heads <- head networks for pooled ROIs e.g. box, mask heads
├─solver       <- optimizer and scheduler builders
├─structures   <- structure classes e.g. Boxes, Instances, etc
└─utils        <- utility modules e.g. visualizer, logger, etc
```

https://medium.com/@hirotoschwert/digging-into-detectron-2-47b2e794fabd
https://medium.com/deepvisionguru/

https://medium.com/deepvisionguru/how-to-embed-detectron2-in-your-computer-vision-project-817f29149461
УСТАНОВКА ПАКЕТОВ

```
!pip install pyyaml==5.1

import torch
TORCH_VERSION = ".".join(torch.__version__.split(".")[:2])
CUDA_VERSION  = torch.__version__.split("+")[-1]
print("torch: ", TORCH_VERSION, "; cuda: ", CUDA_VERSION)
```

после установки необходимо будет перезапустить среду

Если у вас возникли проблемы с установкой, вы можете проверить правильность ссылки установки для вашей версии `torch`, перейдя по ссылке https://detectron2.readthedocs.io/en/latest/tutorials/install.html или используя официальный пример colab: https://colab.research.google.com/drive/16jcaJoc6bCFAQ96jDe2HwtXj7BMD_-m5

ИМПОРТ

```
import detectron2
import torch, torchvision

import numpy as np
import os, json, cv2, random
from google.colab.patches import cv2_imshow

from detectron2        import model_zoo
from detectron2.engine import DefaultPredictor, DefaultTrainer
from detectron2.config import get_cfg
from detectron2.utils.visualizer import Visualizer
from detectron2.data             import MetadataCatalog,
DatasetCatalog
from detectron2.structures       import BoxMode
from detectron2.data.datasets    import register_coco_instances,
load_coco_json
from detectron2.utils.logger     import setup_logger
from detectron2.utils.visualizer import ColorMode
from detectron2.evaluation       import COCOEvaluator,
```

```
inference_on_dataset
from detectron2.data            import build_detection_test_loader
```

Проверка весий

```
print(torch.__version__, torch.cuda.is_available())
!gcc --version
assert torch.__version__.startswith("1.7")
```

```
setup_logger()
```

посмотрим какие есть архитектуры

```
from detectron2.model_zoo.model_zoo import _ModelZooUrls
mz = _ModelZooUrls()
print("\n".join(list(mz.CONFIG_PATH_TO_URL_SUFFIX.keys())))
```

Детекция изображений (Object Detection)

Запустим на тесте - изображении из т.н. coco dataset

```
!wget http://images.cocodataset.org/val2017/000000439715.jpg -q -O
input.jpg
im = cv2.imread("./input.jpg")
cv2_imshow(im)
```

Создадим конфигурацию для detectron2 для модели по умолчанию *DefaultPredictor* для проверки работоспособности модели.

```
cfg = get_cfg()
```

Загрузим кнофигурацию из готовых *model_zoo* например *faster rcnn* на основе энкодера *resnet 50*

```
config =
model_zoo.get_config_file("COCO-Detection/faster_rcnn_R_50_DC5_3x.yaml
")
cfg.merge_from_file(config)
```

Загрузим предобученную модель из набора готовых моделей *model zoo*

```
model_weights      =
model_zoo.get_checkpoint_url("COCO-Detection/faster_rcnn_R_50_DC5_3x.y
aml")

cfg.MODEL.WEIGHTS = model_weights
```

И нам нужно указать порог достоверности для выхода сети

```
cfg.MODEL.ROI_HEADS.SCORE_THRESH_TEST = 0.4
```

загрузим конфигурацию в предиктор

```
predictor = DefaultPredictor(cfg)
```

проверим как работает для загруженного изображения

построем предсказание
```
outputs = predictor(im)
```

Проведем визуализацию предсказания

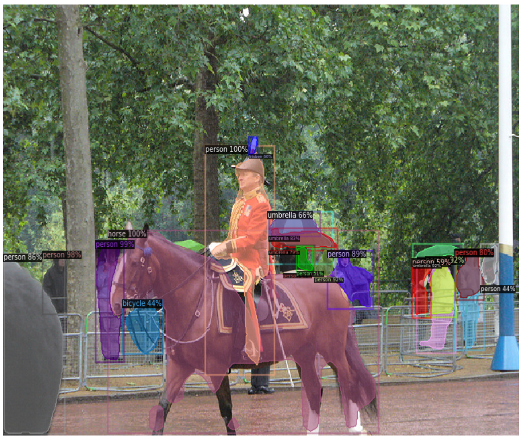
```
v = Visualizer(im[:, :, ::-1],
               MetadataCatalog.get(cfg.DATASETS.TEST[0]),
               scale=1.2)

out = v.draw_instance_predictions(outputs["instances"].to("cpu"))

cv2_imshow(out.get_image()[:, :, ::-1])
```

Экземплярная сегментация (Instace Segmentation)

Теперь проверим тоже самое, но для instance segmentation

```
MODEL = "COCO-InstanceSegmentation/mask_rcnn_R_50_C4_1x.yaml"

cfg           = get_cfg()
model_config  = model_zoo.get_config_file(MODEL)
model_weights =  model_zoo.get_checkpoint_url(MODEL)

cfg.merge_from_file(model_config)
cfg.MODEL.WEIGHTS = model_weights
cfg.MODEL.ROI_HEADS.SCORE_THRESH_TEST = 0.4
predictor = DefaultPredictor(cfg)
outputs   = predictor(im)

v = Visualizer(im[:, :, ::-1],
               MetadataCatalog.get(cfg.DATASETS.TEST[0]),
               scale=1.2)
out = v.draw_instance_predictions(outputs["instances"].to("cpu"))
cv2_imshow(out.get_image()[:, :, ::-1])
```

```
model_final_9243eb.pkl: 144MB [00:06, 20.7MB/s]
```



Сегментация по ключевым точкам (Key Point Segmentation)

```
MODEL = "COCO-Keypoints/keypoint_rcnn_R_50_FPN_3x.yaml"


# Inference with a keypoint detection model
cfg = get_cfg()   # get a fresh new config
cfg.merge_from_file(model_zoo.get_config_file(MODEL))
cfg.MODEL.ROI_HEADS.SCORE_THRESH_TEST = 0.7  # set threshold for this
model
cfg.MODEL.WEIGHTS = model_zoo.get_checkpoint_url(MODEL)
predictor = DefaultPredictor(cfg)
outputs = predictor(im)

v = Visualizer(im[:,:,::1],
MetadataCatalog.get(cfg.DATASETS.TRAIN[0]), scale=1.2)
out = v.draw_instance_predictions(outputs["instances"].to("cpu"))
cv2_imshow(out.get_image()[:, :, ::1])
```

Пан-оптическая сегментация (Panoptic Segmentation)

```
MODEL = "COCO-PanopticSegmentation/panoptic_fpn_R_101_3x.yaml"

cfg = get_cfg()
cfg.merge_from_file(model_zoo.get_config_file(MODEL))
cfg.MODEL.WEIGHTS = model_zoo.get_checkpoint_url(MODEL)

predictor = DefaultPredictor(cfg)

panoptic_seg, segments_info = predictor(im)["panoptic_seg"]
```

```
v = Visualizer(im[:, :, ::-1],
MetadataCatalog.get(cfg.DATASETS.TRAIN[0]), scale=1.2)

out = v.draw_panoptic_seg_predictions(panoptic_seg.to("cpu"),
segments_info)

cv2_imshow(out.get_image()[:, :, ::-1])
```

```
model_final_cafdb1.pkl: 261MB [00:11, 22.7MB/s]
```

Тренировка MASK RCNN для задачи экземплярной сегментации

Рассмотрим задачу тренировки модели из набора, с 1 классом the balloon dataset

Скачаем датасет и разархивируем его

```
# download, decompress the data
!wget
https://github.com/matterport/Mask_RCNN/releases/download/v2.1/balloon
_dataset.zip
!unzip balloon_dataset.zip > /dev/null
```

```
balloon_dataset.zip   0%[                          ]       0  --.-KB/s
^C
replace balloon/train/via_region_data.json? [y]es, [n]o, [A]ll,
[N]one, [r]ename:
```

Теперь надо [зарегистировать датасет](#) - то есть преобразовать его к внутренниму формату детектрона. Для этого используем следующую функцию.

Следуе отметить, что для датасета формата COCO *данная функция ненужна. ЕЕ можно заменить на следующую.*

```
from detectron2.data.datasets import register_coco_instances
register_coco_instances("my_dataset_train", {},
"json_annotation_train.json","path/to/image/dir")
register_coco_instances("my_dataset_val", {},
"json_annotation_val.json", "path/to/image/dir")

from detectron2.structures import BoxMode

def get_balloon_dicts(img_dir):
```

```
    json_file = os.path.join(img_dir, "via_region_data.json")
    with open(json_file) as f:
        imgs_anns = json.load(f)

    dataset_dicts = []
    for idx, v in enumerate(imgs_anns.values()):
        record = {}

        filename = os.path.join(img_dir, v["filename"])
        height, width = cv2.imread(filename).shape[:2]

        record["file_name"] = filename
        record["image_id"] = idx
        record["height"] = height
        record["width"] = width

        annos = v["regions"]
        objs = []
        for _, anno in annos.items():
            assert not anno["region_attributes"]
            anno = anno["shape_attributes"]
            px = anno["all_points_x"]
            py = anno["all_points_y"]
            poly = [(x + 0.5, y + 0.5) for x, y in zip(px, py)]
            poly = [p for x in poly for p in x]

            obj = {
                "bbox": [np.min(px), np.min(py), np.max(px),
np.max(py)],
                "bbox_mode": BoxMode.XYXY_ABS,
                "segmentation": [poly],
                "category_id": 0,
            }
            objs.append(obj)
        record["annotations"] = objs
        dataset_dicts.append(record)
    return dataset_dicts
```

Для двух каталогов train и val проведем регистрацию данных и методанных - их разметки.

!*Следует отметить, что в данном слаучае у нас нет тестового датасета - поэтому мы будем использовать валидационный датасет как тестовый, а валидировать будем на части тренировочного датасета(то есть он будет разбит на две части в рамках процедуры обучения!).*

```
for d in ["train", "val"]:
    try:
        DatasetCatalog.register("balloon_" + d, lambda d=d:
get_balloon_dicts("balloon/" + d))
    except:
         print('Probably data %s have been already registred'%d)
```

```
    MetadataCatalog.get("balloon_" + d).set(thing_classes=["balloon"])
```

```
Probably data train have been already registred
Probably data val have been already registred
```

Теперь проверим что все прошло удачно - для этого загрузим одно случайное изображение и маску для него из тренировочного каталога

Для этого получим данные и методанные для тренировочного датасета

```
balloon_metadata = MetadataCatalog.get("balloon_train")
dataset_dicts    = get_balloon_dicts("balloon/train")
```

теперь возьтем и посмотрим на 2 изображения

```
for d in random.sample(dataset_dicts, 2):

    print(d)

    img = cv2.imread(d["file_name"])

    cv2_imshow(img)

    visualizer = Visualizer(img[:, :, ::-1],
metadata=balloon_metadata, scale=0.1)

    out = visualizer.draw_dataset_dict(d)

    cv2_imshow(out.get_image()[:, :, ::-1])
```

```
Output hidden; open in https://colab.research.google.com to view.
```

Тренировка

Теперь проведем тренировку - дообучение instace segmentataion шариков использованной ранее модели, обученной на датасете *COCO*

[*] Может занять порядка 6 минут для 300 эпох.

Импорт функции тренировщика

```
from detectron2.engine import DefaultTrainer
```

Настройка конфигурации тренировки

- Конфигурация на основе mask_rcnn_R_50_FPN_3x
- датасет *balloon_train*
- число эпох 300
- батч 128 изображений
- число классов 1

```
cfg = get_cfg()
cfg.merge_from_file(model_zoo.get_config_file("COCO-InstanceSegmentati
on/mask_rcnn_R_50_FPN_3x.yaml"))
cfg.DATASETS.TRAIN = ("balloon_train",)
cfg.DATASETS.TEST = ()
```

```
cfg.DATALOADER.NUM_WORKERS = 2
cfg.MODEL.WEIGHTS =
model_zoo.get_checkpoint_url("COCO-InstanceSegmentation/mask_rcnn_R_50
_FPN_3x.yaml")  # Let training initialize from model zoo
cfg.SOLVER.IMS_PER_BATCH = 2
cfg.SOLVER.BASE_LR = 0.00025  # pick a good LR
cfg.SOLVER.MAX_ITER = 300    # 300 iterations seems good enough for
this toy dataset; you will need to train longer for a practical
dataset
cfg.MODEL.ROI_HEADS.BATCH_SIZE_PER_IMAGE = 128   # faster, and good
enough for this toy dataset (default: 512)
cfg.MODEL.ROI_HEADS.NUM_CLASSES = 1  # only has one class (ballon).
(see
https://detectron2.readthedocs.io/tutorials/datasets.html#update-the-c
onfig-for-new-datasets)
```

Процедура тренировки

```
os.makedirs(cfg.OUTPUT_DIR, exist_ok=True)
trainer = DefaultTrainer(cfg)
trainer.resume_or_load(resume=False)
trainer.train()
```

В случае необходимости посмотреть результаты тренировки разкомментируйте следующих код

```
# Look at training curves in tensorboard:
%load_ext tensorboard
%tensorboard --logdir output
```

Оценка результатов обучения нейронной сети

Проведем оценку результатов обучения на валидационном наборе данных.

Для этого создадим объект - предсказатель, в котором используем результаты только что обученной модели.

* Конфигурация предсказателя должна быть основана на конфигурации для тренировки. Но в ней нужно добавить путь к обученным весам и выставить порог принятия решения об объекте (вероятность объекта не менее 70%).

```
cfg.MODEL.WEIGHTS = os.path.join(cfg.OUTPUT_DIR, "model_final.pth")
cfg.MODEL.ROI_HEADS.SCORE_THRESH_TEST = 0.7
predictor = DefaultPredictor(cfg)
```

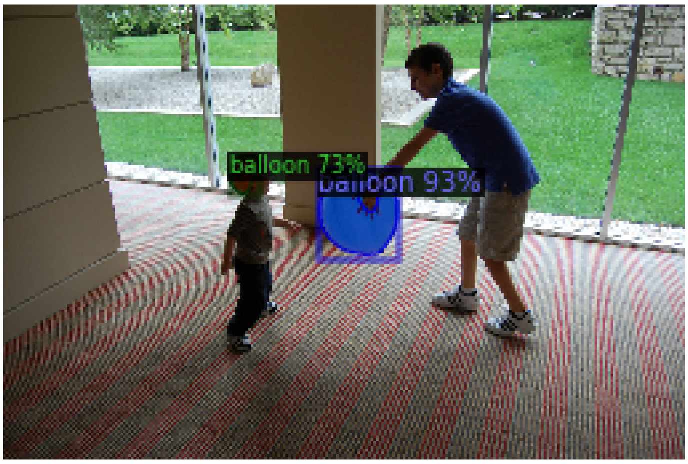
Посмотрим, что получислось

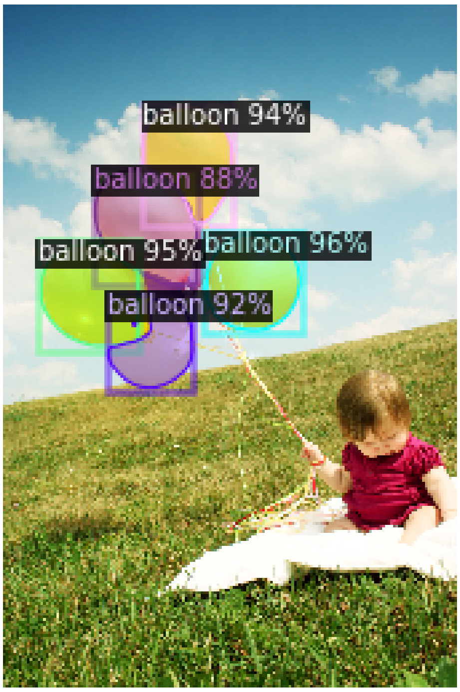
```
from detectron2.utils.visualizer import ColorMode
from detectron2.evaluation import COCOEvaluator, inference_on_dataset
from detectron2.data import build_detection_test_loader
```

Возьмем валидационный датасет

```
dataset_dicts = get_balloon_dicts("balloon/val")

for d in random.sample(dataset_dicts, 3):
    im = cv2.imread(d["file_name"])
```

```
    outputs = predictor(im)  # format is documented at https://detectron2.readthedocs.io/tutorials/models.html#model-output-format
    v = Visualizer(im[:, :, ::-1],
                   metadata=balloon_metadata,
                   scale=0.25,
                   instance_mode=ColorMode.IMAGE_BW   # remove the colors of unsegmented pixels. This option is only available for segmentation models
    )
    out = v.draw_instance_predictions(outputs["instances"].to("cpu"))
    cv2_imshow(out.get_image()[:, :, ::-1])
```





Проведем количественную оценку точности результатов обучения сети. Для этого используем метрику средняя точность (average precision, AP), а также оценим среднюю полноту результатов обучения (average recall, AR).

Создадим объект - оценка точности

```
evaluator  = COCOEvaluator(dataset_name = "balloon_val",
                           tasks        = ("bbox", "segm"),
                           distributed  = False,
                           output_dir   = "./output/")
```

Создадим объект загрузки датасета

```
val_loader = build_detection_test_loader(cfg, "balloon_val")
```

Упражнение 1.

9. Попробуйте обучить модель для задачи обнаружения окон и зданий по ссылке https://github.com/InformationSystemsFreiburg/image_segmentation_japan/raw/master/buildings.zip
10. Попробуйте построить модель обнаружения объектов для набора данных люди на улицах, которую вы можете найти здесь: https://www.cis.upenn.edu/~jshi/ped_html/PennFudanPed.zip *Примечание* набор имеет формат COCO

Лабораторная 7

Работа сопровождается документом в формате ipnb

YOLO V5

YOLOv5 - это семейство моделей обнаружения объектов со сложным масштабированием, подготовленных как дистрибутив на основе PyTorch. Модели предобучены на наборе данных COCO. Дистрибутив Yolo включает простые функции для тренировки, тестирования и валидирования моделей на произвольных наборах данных, а также возможности создания собственных моделей и их экспорта в другие популярные форматы, такие как ONNX, CoreML и TFLite.

Мы настоятельно рекомендуем выполнять данную работу в Google Colab.
Import

```
import cv2
import torch
from PIL import Image
import matplotlib.pyplot as plt
import IPython
from IPython import display
```

Загрузим официальный репозиторий с моделью и установим все требования.

```
%cd /content/
!git clone https://github.com/ultralytics/yolov5
%cd yolov5
!pip install -r requirements.txt
try:
    import yolov5
except:
    pass
else:
    display.clear_output()

print(f"Setup complete. Using torch {torch.__version__}
({torch.cuda.get_device_properties(0).name if
torch.cuda.is_available() else 'CPU'})")
```

```
Setup complete. Using torch 1.10.0+cu111 (CPU)
```

Теперь вы можете проверить, что на вашей виртуальной машине есть репозиторий, как показано ниже.

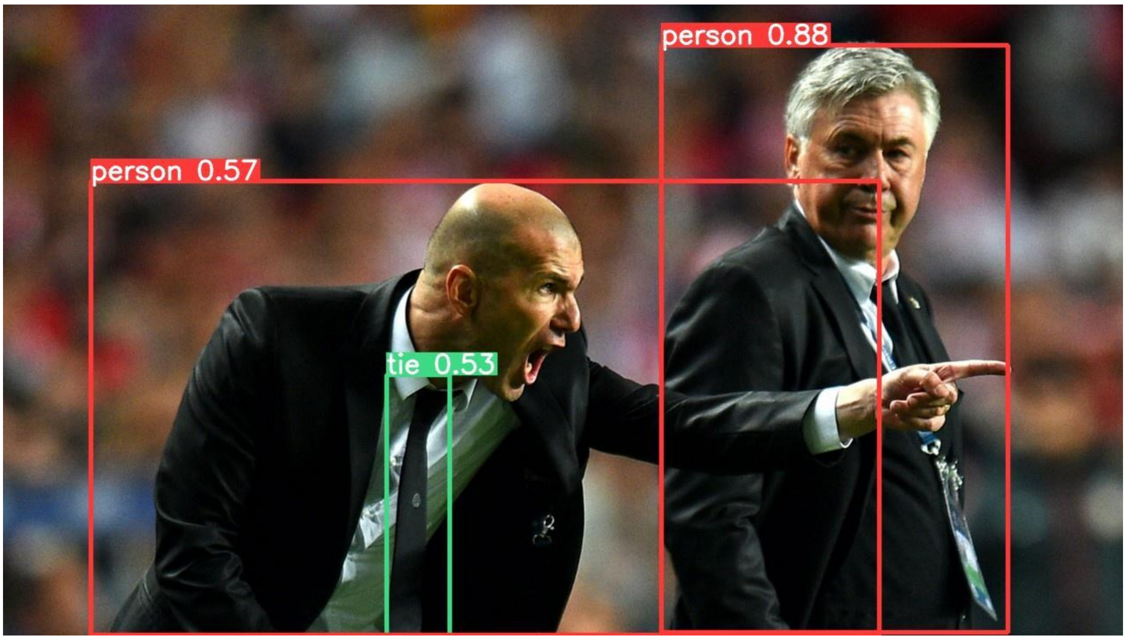
Также мы можем проверить это с помощью команд bash

```
!pwd
!ls
```

```
CONTRIBUTING.md  export.py   README.md         train.py        yolov5
data         hubconf.py  requirements.txt  tutorial.ipynb  yolov5s.pt
detect.py    LICENSE     runs              utils           yolov5x.pt
Dockerfile   models      setup.cfg             val.py
```

Мы можем проверить, что Yolo работает, используя следующий код

```
!rm -r -f /content/yolov5/runs/detect/*

!python detect.py --weights yolov5s.pt --img 640 --conf 0.25 --source
data/images

display.Image(filename='/content/yolov5/runs/detect/exp/zidane.jpg',
width=600)

detect: weights=['yolov5s.pt'], source=data/images, imgsz=[640, 640],
conf_thres=0.25, iou_thres=0.45, max_det=1000, device=,
view_img=False, save_txt=False, save_conf=False, save_crop=False,
nosave=False, classes=None, agnostic_nms=False, augment=False,
visualize=False, update=False, project=runs/detect, name=exp,
exist_ok=False, line_thickness=3, hide_labels=False, hide_conf=False,
half=False, dnn=False
YOLOv5 🚀 v6.0-84-gdef7a0f torch 1.10.0+cu111 CPU

Fusing layers...
Model Summary: 213 layers, 7225885 parameters, 0 gradients
image 1/2 /content/yolov5/data/images/bus.jpg: 640x480 4 persons, 1
bus, Done. (0.283s)
image 2/2 /content/yolov5/data/images/zidane.jpg: 384x640 2 persons, 1
tie, Done. (0.215s)
Speed: 2.2ms pre-process, 248.9ms inference, 1.2ms NMS per image at
shape (1, 3, 640, 640)
Results saved to runs/detect/exp
```



COCO Dataset validation

Во-первых, мы можем проверить, как YOLO будет работать с набором данных COCO. Для тестов COCO воспользуемся скриптом val.py. Скрипт можно вызвать со следующими аргументами: '--data', default=ROOT / 'data/coco128.yaml', 'путь к файлу вида dataset.yaml' '--weights' , nargs = '+', default = ROOT / 'yolov5s.pt', 'model.pt path (s)' '--batch-size' , по умолчанию = 32, 'размер партии' '--imgsz', '--img', '--img-size' ,, по умолчанию = 640, 'размер вывода (в пикселях)' '--conf-thres' , по умолчанию = 0,001, 'порог достоверности' '--iou-thres' , по умолчанию = 0,6, 'Порог NMS IoU' '--task' , default = 'val', 'train, val, test, speed or study' '--device' , default = '', 'cuda device, т.е. 0 или 0,1,2,3 или cpu' '--single-cls' , 'рассматривать как набор данных одного класса' '--augment' , 'расширенный вывод' '--verbose' , 'сообщить MAP по классам' '--save-txt' , 'сохранять результаты в *.txt' '--save-hybrid' , 'сохранить гибридные результаты метки + прогноза в* .txt' '--save-conf' , 'сохранить достоверность в ярлыках --save-txt' '--save-json' , 'сохранить файл результатов COCO-JSON' '--project' , по умолчанию = ROOT/'run/ val', 'save to project / name' '--name' , default = 'exp', 'сохранить в проект / имя' '--exist-ok' , 'существующий проект / имя в порядке, без увеличения' '--half' , 'использовать вывод половинной точности FP16' '--dnn' , 'использовать OpenCV DNN для вывода ONNX'

Для проверки мы будем использовать маленькую модель `yolov5s.pt`
Примечание
Результаты проверки сохраняются в 'run/val/' с увеличивающимися номерами каталога выполнения, то есть 'run/val/ exp2', 'run/val/exp3' и т.д.
Вместо использования полного набора данных COCO мы можем проверить результаты на уменьшенной копии этого набора COCO128.
Предварительно обученные модели загружаются автоматически из последней версии репозитория YOLOv5, вы можете найти ее в `/yolo/models/'.
Наборы данных , доступные для автоматической загрузки, включают: COCO, COCO128, VOC, Argoverse, VisDrone, GlobalWheat, xView, Objects365, SKU-110K .
Вы можете найти всю доступную информацию о текущем состоянии Yolov5, используя страницу информации о релизах : https://github.com/ultralytics/yolov5/Release/
Также вы можете загрузить наборы данных COCO и COCO128 вручную. Например, для COCO128 вы можете использовать следующий код:

```
# Download COCO val
torch.hub.download_url_to_file('https://ultralytics.com/assets/coco128
.zip', 'tmp.zip')
!unzip -q tmp.zip -d ../datasets && rm tmp.zip

!python val.py --weights 'yolov5s.pt' --data coco128.yaml --img 640
--iou 0.65 --half

val: data=/content/yolov5/data/coco128.yaml, weights=['yolov5s.pt'],
batch_size=32, imgsz=640, conf_thres=0.001, iou_thres=0.65, task=val,
device=, single_cls=False, augment=False, verbose=False,
save_txt=False, save_hybrid=False, save_conf=False, save_json=False,
project=runs/val, name=exp, exist_ok=False, half=True, dnn=False
YOLOv5 🚀 v6.0-84-gdef7a0f torch 1.10.0+cu111 CPU


Fusing layers...
Model Summary: 213 layers, 7225885 parameters, 0 gradients
val: Scanning '../datasets/coco128/labels/train2017.cache' images and
labels... 128 found, 0 missing, 2 empty, 0 corrupted: 100% 128/128
[00:00<?, ?it/s]
               Class     Images     Labels          P          R
mAP@.5 mAP@.5:.95: 100% 4/4 [00:49<00:00, 12.44s/it]
                 all        128        929      0.664      0.536
0.619      0.409
Speed: 5.1ms pre-process, 356.5ms inference, 9.2ms NMS per image at
```

```
shape (32, 3, 640, 640)
Results saved to runs/val/exp8
```

Вы можете найти всю информацию о результатах обучения, проверки и обнаружения в каталоге */yolov5/run/*. Например, мы можем увидеть, как выглядят результаты для последнего батча

```
display.Image(filename='/content/yolov5/runs/val/exp/val_batch0_labels
.jpg', width=1024)
```

Тренировка

Для обучения воспользуемся подготовленным скриптом *train.py*. путь начальных весов: *--weights*, по умолчанию = ROOT/'yolov5s.pt'

путь к model.yaml для конфигурации: *--cfg* default = ''
путь к dataset.yaml: *--data*, по умолчанию = ROOT / 'data / coco128.yaml'
путь к конфигурации гиперпараметров: *--hyp*, по умолчанию = ROOT / 'data / hyps / hyp.scratch.yaml'
количество эпох: *--epochs*, по умолчанию = 300
общий размер пакета для всех графических процессоров: *--batch-size* по умолчанию = 16, -1 для автоподбора
тренировка, размер изображения val (в пикселях): *--imgsz*,*--img*, *--img-size*, по умолчанию = 640
использовать взвешенный выбор изображений для обучения: *--image-weights*, action = 'store_true`
обучать данные с несколькими классами как с одним классом: *--single-cls*, action = 'store_true'
сохранить в проект/имя: *--project*, по умолчанию = ROOT/'run/train'
сохранить в проект/имя: *--name*, default = 'exp'
Количество слоев для закрепления. backbone = 10, all = 24: *--freeze*, по умолчанию = 0
Сохранять контрольную точку каждые x эпох (отключено, если <1): *--save-period*, по умолчанию = -1

- Примечание * Вы можете найти остальные параметры в *train.py* в форме, подобном: *parser.add_argument ('- freeze', default = 0, 'Number of Layers to freeze . backbone = 10, all = 24 ')*

Например, мы обучим модель YOLOv5s на наборе данных COCO128 с помощью *--data coco128.yaml*, с предобученными весами *--weights yolov5s.pt* со спецификацией, определенной в файле *--cfg 'yolov5s.yaml*.

Примечание Результаты обучения сохраняются в 'run/train/' с увеличением номера каталогов, например, 'run/train/exp2', 'run/train/exp3' и т. д.

```
# # Tensorboard  (optional)
# %load_ext tensorboard
# %tensorboard --logdir runs/train

# Train YOLOv5s on COCO128 for 3 epochs
!python train.py --img 640 --batch 16 --epochs 3 --data coco128.yaml
--weights yolov5s.pt --cache
```

```
python3: can't open file 'train.py': [Errno 2] No such file or
directory
```

Упражнение 1.

11. Проверьте производительность(точность) дообученной модели на наборе валидационных данных.
12. Найдите и проверьте графические результаты обучения (*results.png*).
13. Проверьте производительность модели Yolov5 nano для набора данных *COCO128*, сравните метрику *mAP @ .5* для модели small и nano
14. Попробуйте обучить модель yolo nano с нуля (без предобченных весов) с минимум 100 эпохами обучения для задачи *COCO128*.
15. Проверьте производительность обученной модели на валидационных данных.

Упражнение 2.

16. Попробуйте обучить модель YOLO small (предобученную) с помощью следующего пользовательского набора данных.

```
! wget
https://raw.githubusercontent.com/shitkov/signature_detector/main/data
set.zip
! распаковать dataset.zip
display.clear_output ()
```

Используйте размер изображения 640, размер батчв 16, количество эпох 100. *Примечание* данные в обучении должны быть указаны как файл *.yaml*, например *'/content/yolov5/dataset/sig.yaml'*

17. Проверьте результаты в для валидационных данных.

Упражнение 3. Продвинутый уровень (не обязательное).

18. Изучите, как обучить YOLO на пользовательских данных, используя, например,

- это руководство https://blog.roboflow.com/how-to-train-yolov5-on-a-custom-dataset/?ref=ultralytics
- или этот блокнот https://colab.research.google.com/github/roboflow-ai/yolov5-custom-training-tutorial/blob/main/yolov5-custom-training.ipynb
- или это неофициальное руководство https://wandb.ai/onlineinference/YOLO/reports/YOLOv5-Object-Detection-on-Windows-Step-By-Step-Tutorial---VmlldzoxMDQwNzk4
- или этот блокнот https://colab.research.google.com/drive/1gDZ2xcTOgR39tGGs-EZ6i3RTs16wmzZQ#scrollTo=SDIhrBF0sPaM

19. Также ознакомьтесь с официальной вики-страницей YOLO https://github.com/ultralytics/yolov5/wiki.

Лабораторная 8

Работа сопровождается документом в формате ipnb

GAN in PyTorch

Вступление

Генеративная состязательная сеть (GAN) позволяет генерировать новые изображения после демонстрации (обуения) реальных примеров.
Цель GAN - научить модель глубокого обучения фиксировать распределения тренировочных данных, чтобы мы могли генерировать новые данные из того же распределения. Сети GAN были изобретены Яном Гудфеллоу в 2014 году. Общая модель GAN состоит из *генератора* ($G(z)$, где z - некоторый стохастический вход (т.е. шумы)) и *дискриминатора* ($ D (x) $, где $ x$ - пример реального изображения).
Задача генератора - создавать «поддельные» изображения, похожие на обучающие. Задача дискриминатора - определить, является ли выходной сигнал действительным или фейком - поддельным изображением (изображение с генератора). Во время обучения генератор постоянно пытается перехитрить дискриминатор, генерируя все более качественные поддельные изображения, в то время как дискриминатор работает над тем, чтобы стать лучше и правильно классифицировать реальные и поддельные изображения. Равновесие этого процесса - когда генератор генерирует идеальные подделки, которые выглядят так, как будто они пришли непосредственно из данных обучения, а дискриминатору остается всегда угадывать с 50% вероятностью, настоящий или фальшивый пример перед ним.
Первоначальная идея GAN заключалась в том, чтобы иметь минимаксную задачу для $D $ и $ G $ ($ D
) − тоестьзадачу, кодадискреминаторпытаетсямаксимизироватьвероятностьтого, чтоонпра
\log D(x)$), а $ G $ пытается минимизировать вероятность того, что $ D $ предсказывает, что его выходные данные являются поддельными ($ \log(1-D (G (z))$). Тогда функция потерь GAN имеет вид \begin{align} \underset{G}{\text{min}} \underset{D}{\text{max}}V(D,G) = \mathbb{E}*{x\sim p*{data}(x)}\big[logD(x)\big] + \mathbb{E}*{z\sim p*{z}(z)}\big[log(1-D(G(z)))\big] \end{align}
Теоретически решение этой проблемы - это ситуация когда $p_g = p_{data}$, и дискриминатор случайным образом угадывает, являются ли входные данные настоящими или поддельными. Однако теория сходимости GAN все еще активно исследуется, и в действительности модели не всегда тренируйтесь до этого момента.

> *Примечание*. Мы будем использовать упрощенную модель из-за высокой вычислительной сложности полных моделей GAN. Данный практикум даст вам только объяснение общих принципо реализации GAN и того, как и почему работает эта модель.
>
> Ради экономии времени мы рекомендуем вам использовать графический процессор. Если вы работаете в Google Colab, вы можете включить его в меню мультимедиа.

! $^$ *Примечание*:

Если вы используете Google Colab, можно использовать CPU или GPU.
Для изменения **среды выполнения** выберите **Изменить тип среды выполнения** в соответствующем меню. После экспериментов не забудьте вернуть тип среды на None (CPU). Google предоставляет графический процессор только временно.

```
 import torch
import torch.nn as nn
import torch.nn.parallel
```

```
import torch.backends.cudnn as cudnn
import torch.optim as optim
import torch.utils.data
import torchvision.datasets as datasets
import torchvision.transforms as transforms
import torchvision.utils as vutils

from torch.utils.data import DataLoader

import os
import random

import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation
%matplotlib inline
from IPython.display import HTML

"""
Determine if any GPUs are available
"""
DEVICE = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

DEVICE
```

```
device(type='cpu')
```

```
import random
# Set random seed for reproducibility
manualSeed = 999
#manualSeed = random.randint(1, 10000) # use if you want new results
print("Random Seed: ", manualSeed)
random.seed(manualSeed)
torch.manual_seed(manualSeed)
```

```
Random Seed:  999

<torch._C.Generator at 0x25dbaadf5d0>
```

Набор данных.

```
BATCH_SIZE = 128
NOISE_SIZE = 128 #Z_NOISES
IMAGE_SIZE = 28  #IMAGE WIDHT AND HEIGHT

transform=transforms.Compose([transforms.ToTensor(),
                              transforms.Normalize((0.5,), (0.5,)),
                             ])

train_set = datasets.MNIST('mnist/',
                           train     = True,
                           download  = True,
                           transform = transform)
```

```
dataloader = torch.utils.data.DataLoader(train_set,
                                         batch_size=BATCH_SIZE,
                                         shuffle=True,
                                         drop_last=True)
```

Посмотрим на некоторые из примеров изображений

```
x_batch = next(iter(dataloader))

plt.figure(figsize=(8,8))

plt.title("Training Images")
plt.imshow(np.transpose(vutils.make_grid(x_batch[0].to(DEVICE)[:16],
                                         nrow = 4,
                                         normalize=True).cpu(),
                        (1,2,0)))
plt.show()
```

Реализация

Инициалиазция весовых параметров

```
# custom weights initialization called on G and D
def weights_init(m):
    classname = m.__class__.__name__
    if classname.find('Conv') != -1:
        nn.init.normal_(m.weight.data, 0.0, 0.02)
    elif classname.find('BatchNorm') != -1:
        nn.init.normal_(m.weight.data, 1.0, 0.02)
        nn.init.constant_(m.bias.data, 0)
```

Архитектура модели

Генератор $G $ отображает вектор скрытого пространства ($ z$) в пространство данных (форма изображения). Дискриминатор D представляет собой сеть бинарной классификации, которая принимает изображение на входе и выводит вероятность что входное изображение настоящее на выходе

```
def vanilla_block(in_feat,
                  out_feat,
                  normalize=True,
                  activation=None):
```

```python
        layers = []

        layers.append(nn.Linear(in_feat, out_feat))

        if normalize:
            layers.append(nn.BatchNorm1d(out_feat))

        if activation:
            layers.append(activation)
        else:
            layers.append(nn.LeakyReLU(0.2))

        return layers


class Generator(nn.Module):

    def __init__(self):

        super().__init__()

        self.net = nn.Sequential(
            *vanilla_block(NOISE_SIZE, 256),
            *vanilla_block(256, 512),
            *vanilla_block(512, 1024),
            *vanilla_block(1024, IMAGE_SIZE**2,
                           normalize=False,
                           activation=nn.Tanh())
        )

    def forward(self, latent_vector_batch):
        x = self.net(latent_vector_batch)
        # un-flatten (N, 1, 28, 28) shape for MNIST
        return x.view(-1, 1, IMAGE_SIZE, IMAGE_SIZE)


class Discriminator(nn.Module):

    def __init__(self):
        super().__init__()

        self.net = nn.Sequential(
            *vanilla_block(IMAGE_SIZE**2, 512, normalize=False),
            *vanilla_block(512, 256, normalize=False),
            *vanilla_block(256, 1,
                           normalize=False,
                           activation=nn.Sigmoid())
        )

    def forward(self, img_batch):
        # flatten from (N,1,H,W) into (N, HxW)
```

```
        x = img_batch.view(img_batch.shape[0], -1)
        return self.net(x)
```

Теперь мы можем создать экземпляр генератора и дискриминатора и применить функцию *weights_init*. Посмотрите на модель, чтобы увидеть, как она устроена.

```
# Create the generator
G = Generator().to(DEVICE)

# # Handle multi-gpu if desired
# if (DEVICE.type == 'cuda') and (ngpu > 1):
#     G = nn.DataParallel(G, list(range(ngpu)))

# Apply the weights_init function to randomly initialize all weights
G.apply(weights_init)

# Print the model
print(G)

Generator(
  (net): Sequential(
    (0): Linear(in_features=128, out_features=256, bias=True)
    (1): BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (2): LeakyReLU(negative_slope=0.2)
    (3): Linear(in_features=256, out_features=512, bias=True)
    (4): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (5): LeakyReLU(negative_slope=0.2)
    (6): Linear(in_features=512, out_features=1024, bias=True)
    (7): BatchNorm1d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (8): LeakyReLU(negative_slope=0.2)
    (9): Linear(in_features=1024, out_features=784, bias=True)
    (10): Tanh()
  )
)

# Create the Discriminator
D = Discriminator().to(DEVICE)

# # Handle multi-gpu if desired
# if (DEVICE.type == 'cuda') and (ngpu > 1):
#     D = nn.DataParallel(D, list(range(ngpu)))

# Apply the weights_init function to randomly initialize all weights
D.apply(weights_init)

# Print the model
print(D)

Discriminator(
  (net): Sequential(
```

```
    (0): Linear(in_features=784, out_features=512, bias=True)
    (1): LeakyReLU(negative_slope=0.2)
    (2): Linear(in_features=512, out_features=256, bias=True)
    (3): LeakyReLU(negative_slope=0.2)
    (4): Linear(in_features=256, out_features=1, bias=True)
    (5): Sigmoid()
  )
)
```

Функции потерь и оптимизация

Мы можем указать, как изучать $D $ и $ G$ с помощью функций потерь и оптимизатора. Мы будем использовать функцию бинарной взаимной энтропии, которая определена в PyTorch как: \begin{align} \ell(x, y) = L = {l_1,\dots,l_N}^\top, \quad l_n = - \left[y_n \cdot \log x_n + (1 - y_n) \cdot \log (1 - x_n) \right] \end{align}

> *Примечание*. Эта функция позволяет рассчитывать как $log(D(x))$ так и $log(1 - D[G(z)])$ путем спецификации y и x.

Также нам нужно определить реальное изображение как метку "1" и фальшивку как метку "0". Наконец, мы настроим отдельные оптимизаторы для $D $ и $ G$.

```
# Initialize BCELoss function
criterion = nn.BCELoss()
# Learning rate for optimizers
lr = 0.0002

# Beta hyperparams for Adam optimizers
beta1 = 0.5
beta2 = 0.999

# Setup Adam optimizers for both G and D
optimizerD = optim.Adam(D.parameters(), lr=lr, betas=(beta1, beta2))
optimizerG = optim.Adam(G.parameters(), lr=lr, betas=(beta1, beta2))
```

Для единообразция теста давайте создадим батч скрытых векторов. Мы будем использовать данный батч для визуализации.

```
fixed_noise = torch.randn(16, NOISE_SIZE, device=DEVICE)

def generate_samples(G, z_noise = fixed_noise):
    #instead of G.eval() due to recomendations
    #of using same config for batchnorm and droput
    #in the eval for GAN
    with torch.no_grad():
        z = G(z_noise).detach().cpu()
    return z
```

Установим метки батчей

```
labels_x = torch.ones( BATCH_SIZE).to(DEVICE) # Discriminator Label to
real
labels_z = torch.zeros(BATCH_SIZE).to(DEVICE) # Discriminator Label to
fake
```

Тренировка

```
# Training Loop
EPOCHS = 10

# Lists to keep track of progress
img_list = []
G_losses = []
D_losses = []

print("Starting Training Loop...")
# For each epoch
for epoch in range(EPOCHS):

    # For each batch in the dataloader
    for idx, (images,_) in enumerate(dataloader):
        ############################
        # 1. Train D network: maximize log(D(x)) + log(1 - D(G(z)))
        D.zero_grad()
        G.zero_grad()

        ###########################
        # 1 Train D network with real images

        # 1.1 Format batch and labels
        x = images.to(DEVICE)
        b_size = x.size(0) # to train if not full batch

        # 1.2 Forward pass real batch through D
        output = D(x).view(-1)

        # 1.3 Calculate loss on real batch
        Loss_D_x = criterion(output, labels_x[:b_size])

        # 1.4 Calculate gradients for D (backward pass)
        Loss_D_x.backward()

        ####################################
        # 2 Train with fake batch
        # 2.1 Generate batch of latent vectors
        z_noise = torch.randn(b_size, NOISE_SIZE, device=DEVICE)

        # 2.2 Generate fake image batch with G
        z = G(z_noise)

        # 2.3 Classify fake images batch with D(G(z_noise))
        output = D(z.detach()).view(-1)

        # 2.4 Calculate D loss on the fake batch
        Loss_D_z = criterion(output, labels_z[:b_size])

        # 2.5 Calculate the gradients for fake batch,
```

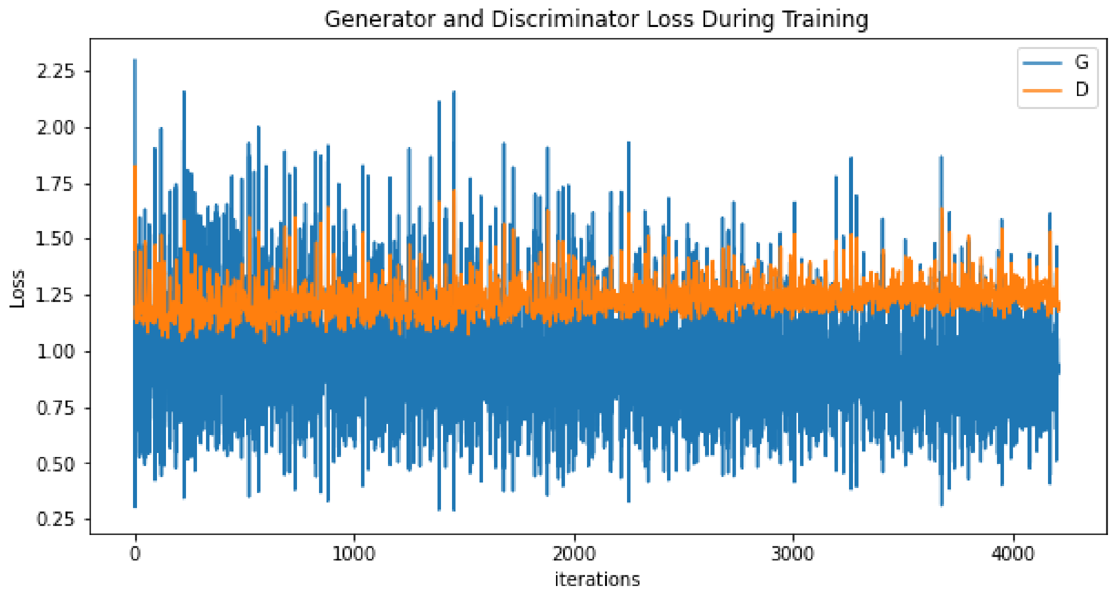
```
        Loss_D_z.backward()

        ########################################
        # 2.7 Compute error of D as sum over the fake and the real batches
        Loss_D = Loss_D_x + Loss_D_z

        # 2.8 Update Discrimenator weights
        optimizerD.step()

        ############################
        # 3. Update G network: maximize log(D(G(z)))

        ###########################
        # 3.1 Since updating D, another forward pass of fake batch through D
        output = D(z).view(-1)

        # 3.2 Calculate G's loss based on this output
        Loss_G = criterion(output, labels_x[:b_size])

        # 3.3 Calculate gradients for G
        Loss_G.backward()

        # 3.4 Update G
        optimizerG.step()

        ########################################
        # 4. Auxiliary part

        # 4.1 Output training stats
        if idx % 50 == 0:
            print("""[%d/%d][%d/%d]\t Loss_D:%.4f\t Loss_G:%.4f\t"""
                  % (epoch, EPOCHS,
                     idx,len(dataloader),
                     Loss_D.item(),
                     Loss_G.item()))

        # 4.2 Save Losses for plotting later
        G_losses.append(Loss_G.item())
        D_losses.append(Loss_D.item())

        # 4.3 Check how the generator is doing by saving G's output on fixed_noise
        if (idx % 500 == 0) or (
            (epoch == EPOCHS-1) and (idx == len(dataloader)-1)
        ):
            z = generate_samples(G)
            img_list.append(vutils.make_grid(z, padding=2, normalize=True))
```

```
Starting Training Loop...
[0/10][0/468]     Loss_D:1.2405    Loss_G:1.1543
[4/10][100/468]   Loss_D:1.2635    Loss_G:0.9093
[4/10][150/468]   Loss_D:1.3173    Loss_G:0.6138
[9/10][250/468]   Loss_D:1.3816    Loss_G:1.1371
[9/10][300/468]   Loss_D:1.2596    Loss_G:0.8510
[9/10][350/468]   Loss_D:1.2474    Loss_G:0.9508
[9/10][400/468]   Loss_D:1.3004    Loss_G:0.6113
[9/10][450/468]   Loss_D:1.3673    Loss_G:1.4952
```

Оценка результатов

Потери по итерациям обучения Ниже приведен график потерь D и G в зависимости от итераций обучения.

```
plt.figure(figsize=(10,5))
plt.title("Generator and Discriminator Loss During Training")
plt.plot(G_losses,label="G")
plt.plot(D_losses,label="D")
plt.xlabel("iterations")
plt.ylabel("Loss")
plt.legend()
plt.show()
```

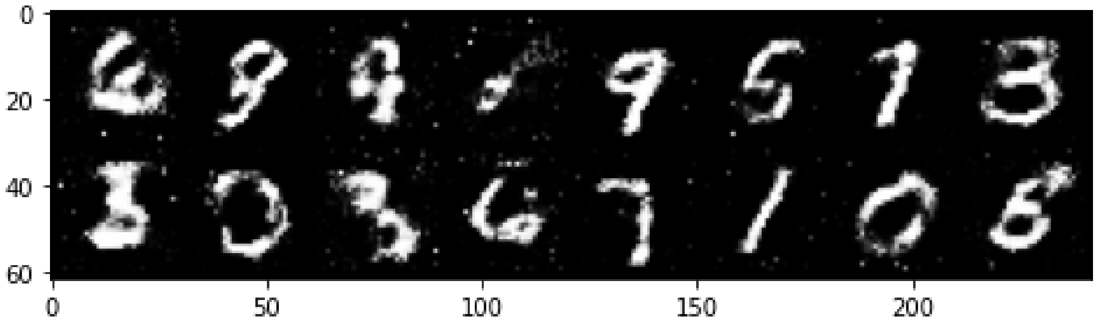


Визуализация прогресса генератора Мы можем визуализировать процесс обучения G с помощью анимации. Нажмите кнопку воспроизведения, чтобы начать анимацию.

```
#%%capture
fig = plt.figure(figsize=(8,8))

ims = [[plt.imshow(np.transpose(i,(1,2,0)), animated=True)] for i in
img_list]

ani = animation.ArtistAnimation(fig, ims, interval=1000,
repeat_delay=1000, blit=True)
```

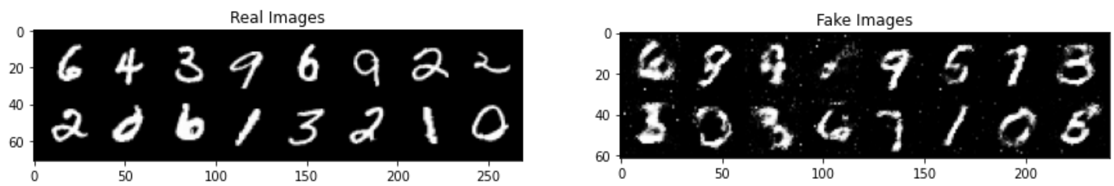
```
HTML(ani.to_jshtml())
```

```
<IPython.core.display.HTML object>
```



Настоящие изображения и поддельные изображения Наконец, давайте посмотрим на несколько реальных и поддельных изображений.

```
# Grab a batch of real images from the dataloader
x_batch = next(iter(dataloader))

# Plot the real images
plt.figure(figsize=(15,15))
plt.subplot(1,2,1)

plt.title("Real Images")
plt.imshow(np.transpose(vutils.make_grid(x_batch[0].to(DEVICE)[:16],
                                         nrow = 8,
                                         padding=5,
                                         normalize=True).cpu(),
                        (1,2,0)))

# Plot the fake images from the last epoch
plt.subplot(1,2,2)

plt.title("Fake Images")
plt.imshow(np.transpose(img_list[-1],(1,2,0)))
plt.show()
```




```
def save_checkpoint(state, file_name='checkpoint.pth.tar'):
    torch.save(state, file_name)

# Saving params.
save_checkpoint({'epoch': epoch + 1, 'state_dict':D.state_dict(),
'optimizer' : optimizerD.state_dict()}, 'D.pth.tar')
save_checkpoint({'epoch': epoch + 1, 'state_dict':G.state_dict(),
'optimizer' : optimizerG.state_dict()}, 'G.pth.tar')
```

Упражнение 1

1. Попробуйте обучить GAN работе с набором данных Fashion MNIST (*datasets.FashionMNIST ()*).
2. Попробуйте создать GAN на основе сверток вместо текущей версии с полносвязными слоями.

CONDITIONAL И INFO GAN

Основная проблема показанного выше обычного GAN - это высокая вероятность переобучения из-за отсутствия какой-либо дополнительной информации об изображении. Это позволяет использовать GAN без учителя. Однако, если у вас есть нужная информация, вы можете использовать ее для повышения производительности GAN. Для этого можно применить два основных подхода. Условный (conditional) GAN - этот подход позволяет учесть требуемый класс изображения, используя дополнительный вектор объедененный (конкатинированный) с шумовым (скрытым) пространством GAN. Info GAN позволяет прогнозировать класс изображения (дискретное распределение) и его конкретные параметры (распределения) в дополнение к принятию решений о реальном/фальшивом изображении. Другими словами, InfoGan разделяет входные данные генератора на две части: обычный вектор шума и новый вектор «скрытого кода». Затем коды становятся значимыми, максимизируя взаимную информацию между кодом и выходом генератора. Эту взаимную информацию можно применить в качестве регуляризационного фактора в функции потерь. \begin{align} \underset{G}{\text{min}} \underset{D}{\text{max}}V(D,G) = \mathbb{E}*{x\sim p*{data}(x)}\big[logD(x)\big] + \mathbb{E}*{z\sim p*{z}(z)}\big[log(1-D(G(z,c)))\big] - \lambda Q(c,G(z,c)), \end{align} где $Q(c, G(z, c))$ это дополнительная сеть для классификации.

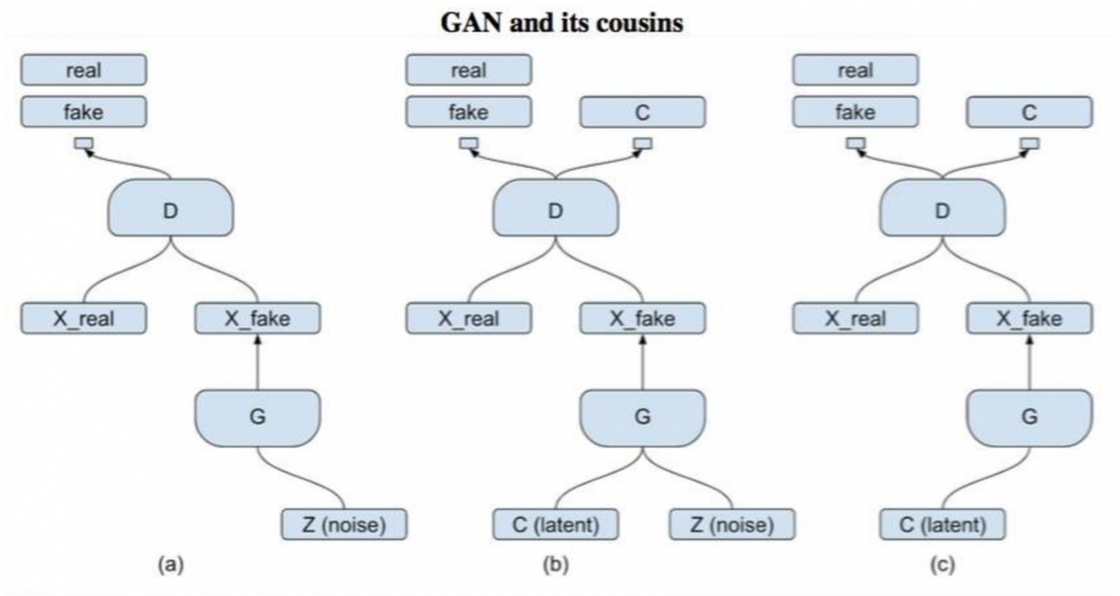


Figure 1: Graphical representation of different GAN configurations: (a) Vanilla GAN; (b) InfoGAN; (c) InfoGAN w/ only latent codes. The style of these diagrams was inspired by [OOS16].

Упражнение 2.

Изучите примеры условного (Conditional) и информационного(Info) GAN ниже.

Добавьте комментарии к ним о том, как они работают, и где их можно или нельзя применять.

```
Conditional GAN

CONDITION_SIZE = 10

def to_onehot(x):
    assert isinstance(x, int) or isinstance(x, (torch.LongTensor,
torch.cuda.LongTensor))

    if isinstance(x, int):
        c = torch.zeros(1, CONDITION_SIZE).long()
        c[0][x] = 1

    else:
        x = x.cpu()
        c = torch.LongTensor(x.size(0), CONDITION_SIZE)
        c.zero_()
        c.scatter_(1, x, 1) # dim, index, src value
    return c

fixed_noise = torch.randn(16, NOISE_SIZE, device=DEVICE)
fixed_c     = torch.zeros([16, CONDITION_SIZE]).to(DEVICE)

def generate_samples(G, z_noise = fixed_noise):
    #instead of G.eval() due to recomendations
    #of using same config for batchnorm and droput
    #in the eval for GAN
    with torch.no_grad():
        z = G(z_noise,fixed_c).detach().cpu()
    return z

class Discriminator(nn.Module):

    def __init__(self):
        super().__init__()

        self.net = nn.Sequential(
            *vanilla_block(IMAGE_SIZE**2+CONDITION_SIZE, 512,
normalize=False),
            *vanilla_block(512, 256, normalize=False),
            *vanilla_block(256, 1,
                           normalize=False,
                           activation=nn.Sigmoid())
        )

    def forward(self, img_batch, condition):

        # flatten from (N,1,H,W) into (N, HxW)
        x = img_batch.view(img_batch.shape[0], -1)
```

```
        # flatten condition into (N, c)
        c = condition.view(condition.size(0), -1).float()
        # create new vector
        # v: [input, label] concatenated vector
        v = torch.cat((x, c), 1)
        return self.net(v)

class Generator(nn.Module):

    def __init__(self):

        super().__init__()

        self.net = nn.Sequential(
            *vanilla_block(NOISE_SIZE+CONDITION_SIZE, 256),
            *vanilla_block(256, 512),
            *vanilla_block(512, 1024),
            *vanilla_block(1024, IMAGE_SIZE**2,
                           normalize=False,
                           activation=nn.Tanh())
        )

    def forward(self, latent_vector_batch, condition):
        # check if input vector
        y = latent_vector_batch.view(latent_vector_batch.size(0), -1)
        # check if condition vector
        c = condition.view(condition.size(0), -1).float()
        # v: [input, label] concatenated vector
        v = torch.cat((y, c), 1)
        # make forward pass
        x = self.net(v)
        # un-flatten (N, 1, 28, 28) shape for MNIST
        return x.view(-1, 1, IMAGE_SIZE, IMAGE_SIZE)

# Create the Discriminator
D = Discriminator().to(DEVICE)
G = Generator().to(DEVICE)

# Apply the weights_init function to randomly initialize all weights
D.apply(weights_init)
G.apply(weights_init)

# Print the model
print(D)
print(G)

Discriminator(
  (net): Sequential(
    (0): Linear(in_features=794, out_features=512, bias=True)
    (1): LeakyReLU(negative_slope=0.2)
    (2): Linear(in_features=512, out_features=256, bias=True)
    (3): LeakyReLU(negative_slope=0.2)
```

```
    (4): Linear(in_features=256, out_features=1, bias=True)
    (5): Sigmoid()
  )
)
Generator(
  (net): Sequential(
    (0): Linear(in_features=138, out_features=256, bias=True)
    (1): BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (2): LeakyReLU(negative_slope=0.2)
    (3): Linear(in_features=256, out_features=512, bias=True)
    (4): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (5): LeakyReLU(negative_slope=0.2)
    (6): Linear(in_features=512, out_features=1024, bias=True)
    (7): BatchNorm1d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (8): LeakyReLU(negative_slope=0.2)
    (9): Linear(in_features=1024, out_features=784, bias=True)
    (10): Tanh()
  )
)

labels_x = labels_x.view(-1,1)
labels_z = labels_z.view(-1,1)

# Lists to keep track of progress
G_losses = []
D_losses = []

for epoch in range(EPOCHS):
    for idx, (images, labels) in enumerate(dataloader):
        # Training Discriminator
        x = images.to(DEVICE)
        y = labels.view(BATCH_SIZE, 1)
        y = to_onehot(y).to(DEVICE)

        x_outputs = D(x, y)
        Loss_D_x = criterion(x_outputs, labels_x)

        z_noise = torch.randn(BATCH_SIZE, NOISE_SIZE).to(DEVICE)
        z_outputs = D(G(z_noise, y), y)
        Loss_D_z = criterion(z_outputs, labels_z)
        Loss_D = Loss_D_x + Loss_D_z

        D.zero_grad()
        Loss_D.backward()
        optimizerD.step()


        # Training Generator
        z = torch.randn(BATCH_SIZE, NOISE_SIZE).to(DEVICE)
```

```python
        z_outputs = D(G(z, y), y)
        Loss_G = criterion(z_outputs, labels_x)

        G.zero_grad()
        Loss_G.backward()
        optimizerG.step()

        ########################################
        # 4. Auxiliary part

        # 4.1 Output training stats
        if idx % 50 == 0:
            print("""[%d/%d][%d/%d]\t Loss_D:%.4f\t Loss_G:%.4f\t"""
                  % (epoch, EPOCHS,
                     idx,len(dataloader),
                     Loss_D.item(),
                     Loss_G.item()))

        # 4.2 Save Losses for plotting later
        G_losses.append(Loss_G.item())
        D_losses.append(Loss_D.item())
```

```
[0/10][0/468]     Loss_D:1.3500    Loss_G:0.6701
[0/10][50/468]    Loss_D:1.3505    Loss_G:0.6703
[9/10][400/468]   Loss_D:1.3542    Loss_G:0.6685
[9/10][450/468]   Loss_D:1.3508    Loss_G:0.6696
```

```python
plt.figure(figsize=(10,5))
plt.title("Generator and Discriminator Loss During Training")
plt.plot(G_losses,label="G")
plt.plot(D_losses,label="D")
plt.xlabel("iterations")
plt.ylabel("Loss")
plt.legend()
plt.show()

# Grab a batch of real images from the dataloader
x_batch = next(iter(dataloader))

# Plot the real images
plt.figure(figsize=(15,15))
plt.subplot(1,2,1)

plt.title("Real Images")
plt.imshow(np.transpose(vutils.make_grid(x_batch[0].to(DEVICE)[:16],
                                         nrow = 8,
                                         padding=5,
                                         normalize=True).cpu(),
                        (1,2,0)))


# Plot the fake images from the last epoch
```

```
plt.subplot(1,2,2)

plt.title("Fake Images")
plt.imshow(np.transpose(,(1,2,0)))
plt.show()
```

Info GAN

```
class Discriminator(nn.Module):

    def __init__(self):
        super().__init__()

        self.net = nn.Sequential(
            *vanilla_block(IMAGE_SIZE**2, 512, normalize=False),
            *vanilla_block(512, 256, normalize=False),
        )

        self.last = nn.Sequential(
             *vanilla_block(256, 1,
                           normalize=False,
                           activation=nn.Sigmoid())
        )

    def forward(self, img):
        x = img.view(img.shape[0], -1)

        # return with top layer features for Q
        features = self.net(x)

        y = self.last(features) # Real / Fake

        return y, features

CODE_SIZE = 12

class Generator(nn.Module):

    def __init__(self):

        super().__init__()

        self.net = nn.Sequential(
            *vanilla_block(NOISE_SIZE+CODE_SIZE, 256),
            *vanilla_block(256, 512),
            *vanilla_block(512, 1024),
            *vanilla_block(1024, IMAGE_SIZE**2,
                           normalize=False,
                           activation=nn.Tanh())
        )
```

```python
    def forward(self, latent_vector, code):

        # check if input vector
        y = latent_vector.view(latent_vector.size(0), -1)

        # check if code vector
        c = code.view(code.size(0), -1).float()

        # v: [input, label] concatenated vector
        v = torch.cat((y, c), 1)

        # make forward pass
        x = self.net(v)

        # un-flatten (N, 1, 28, 28) shape for MNIST
        return x.view(-1, 1, IMAGE_SIZE, IMAGE_SIZE)

CLASSES = 10

class Qrator(nn.Module):
    """
        Regularization Network for increasing Mutual Information
    """
    def __init__(self):
        super(Qrator, self).__init__()
        self.fc = nn.Sequential(
                    *vanilla_block(256, 128),
                     nn.Linear(128, CLASSES+4),
               )

    def forward(self, x):
        # Seperate code
        c = self.fc(x)
        c_discrete = torch.softmax(c[:, :CLASSES], dim=-1) # Digit
Label {0~9}
        c_mu = c[:, 10:12] # mu & var of Rotation & Thickness
        c_var = c[:, 12:14].exp() # mu & var of Rotation & Thickness
        return c_discrete, c_mu, c_var

D = Discriminator().to(DEVICE)
G = Generator().to(DEVICE)
Q = Qrator().to(DEVICE)

def sample_noise(n_c_discrete, n_c_continuous, label=None):

    z = torch.randn(BATCH_SIZE, NOISE_SIZE).to(DEVICE)

    if label:
        c_discrete = to_onehot(label).to(DEVICE) # (B,10)

    else:
        c_discrete = to_onehot(torch.LongTensor(BATCH_SIZE,
```

```
1).random_(0, n_c_discrete)).to(DEVICE) # (B,10)

    c_continuous = torch.zeros(BATCH_SIZE,
n_c_continuous).uniform_(-1, 1).to(DEVICE) # (B,2)

    c = torch.cat((c_discrete.float(), c_continuous), 1)

    return z, c

bce_loss = nn.BCELoss()
ce_loss  = nn.CrossEntropyLoss()

def log_gaussian(c, mu, var):
    """
        criterion for Q(condition classifier)
    """
    return -((c - mu)**2)/(2*var+1e-8) -
0.5*torch.log(2*np.pi*var+1e-8)

n_c_discrete, n_c_continuous = CLASSES, 2

G_losses = []
D_losses = []
GnQ_losses = []

for epoch in range(EPOCHS):
    for idx, (images, labels) in enumerate(dataloader):

        labels = labels.view(BATCH_SIZE, 1)

        # Training Discriminator
        x = images.to(DEVICE)
        x_outputs, _, = D(x)
        D_x_loss = bce_loss(x_outputs, labels_x)

        z, c = sample_noise(n_c_discrete, n_c_continuous,
label=labels)
        z_outputs, _, = D(G(z, c))
        D_z_loss = bce_loss(z_outputs, labels_z)
        D_loss = D_x_loss + D_z_loss

        optimizerD.zero_grad()
        D_loss.backward()
        optimizerD.step()

        # Training Generator
        z, c = sample_noise(n_c_discrete, n_c_continuous,
label=labels)
        c_discrete_label = torch.max(c[:, :-2], 1)[1].view(-1, 1)

        z_outputs, features = D(G(z, c))
        c_discrete_out, cc_mu, cc_var = Q(features)
```

```
        G_loss = bce_loss(z_outputs, labels_x)
        Q_loss_discrete = ce_loss(c_discrete_out,
c_discrete_label.view(-1))

        Q_loss_continuous = -torch.mean(torch.sum(log_gaussian(c[:,
-2:], cc_mu, cc_var), 1)) # N(x | mu,var) -> (B, 2) -> (,1)

        mutual_info_loss = Q_loss_discrete + Q_loss_continuous*0.1

        GnQ_loss = G_loss + mutual_info_loss

        optimizerG.zero_grad()
        GnQ_loss.backward()
        optimizerG.step()

        # Auxiliary part
        if idx % 50 == 0:
            print("""[%d/%d][%d/%d]\t Loss_D:%.4f\t Loss_G:%.4f\t"""
                  % (epoch, EPOCHS,
                     idx,len(dataloader),
                     G_loss.item(),
                     D_loss.item()))

        # 4.2 Save Losses for plotting later
        G_losses.append(G_loss.item())
        D_losses.append(D_loss.item())
        GnQ_losses.append(GnQ_loss.item())
```

```
[0/10][0/468]     Loss_D:0.6709    Loss_G:1.4219

[9/10][450/468]   Loss_D:0.6702    Loss_G:1.4204
```

Список примерных тестовых заданий для зачета:

1. Выберите <u>неверное</u> утверждение касательно причин популярности сверточных нейронных сетей:
    a. Возможность автоматического отбора признаков.
    b. Высокая степень пере-использования весов (эффект памяти).
    c. Сниженное число параметров по сравнению с полно-связными сетями.
2. Выберите <u>верное</u> утверждение касательно причин популярности глубоких нейронных сетей в компьютерном зрении:
    a. Возможность автоматического отбора признаков.
    b. Возможность переобучения в ходе работы.
    c. Сниженное число параметров по сравнению с другими методами машинного обучения.
3. Выберите <u>верное</u> утверждение касательно причин популярности глубоких сверточных нейронных сетей в компьютерном зрении:
    a. Высокая степень пере-использования весов (эффект памяти).
    b. Возможность переобучения в ходе работы.
    c. Сниженное число параметров по сравнению с полно-связными сетями.
4. Выберите <u>неверное</u> утверждение касательно причин популярности глубоких нейронных сетей в компьютерном зрении:
    a. Возможность автоматического отбора признаков.
    b. Возможность переобучения в ходе работы.
    c. Высокая обобщающая способность.

5. Выберите <u>верное</u> утверждение касательно особенностей двухмерной свертки:
    a. Входные данные должны иметь размерность 2.
    b. Каждое ядро свертки должно быть трехмерным.
    c. Каждое ядро производит заданное количество карт признаков.
6. Выберите <u>верное</u> утверждение касательно особенностей двухмерной свертки:
    a. Входные данные должны быть вектором.
    b. Каждое ядро свертки должно быть трехмерным.
    c. Каждое ядро свертки должно быть четырехмерным.
7. Выберите <u>неверное</u> утверждение касательно особенностей двухмерной свертки:
    a. **Входные данные должны быть вектором**.
    b. Каждое ядро свертки должно быть трехмерным.
    c. Входные данные могут быть трехмерными.
8. Выберите <u>неверное</u> утверждение касательно особенностей двухмерной свертки:
    a. Входные данные могут быть трехмерными.
    b. Каждое ядро свертки должно быть трехмерным.
    c. Каждое ядро свертки должно быть четырехмерным.

9. Выберите верное утверждение касательно особенностей двухмерной свертки:
    a. Каскадная свертка — это последовательное соединение горизонтального и вертикального прямоугольных ядер.
    b. Групповая свертка позволяет расширить рецептивное поле.
    c. **Расширенная свертка увеличивает рецептивное поле.**
10. Выберите неверное утверждение касательно особенностей двухмерной свертки:
    a. Пространственно-разряженная (разделенная) свертка — это последовательное соединение горизонтального и вертикального прямоугольных ядер.
    b. **Групповая свертка позволяет расширить рецептивное поле.**
    c. Расширенная свертка увеличивает рецептивное поле.
11. Выберите верное утверждение касательно особенностей двухмерной свертки:
    a. Каскадная свертка увеличивает рецептивное поле.
    b. Групповая свертка позволяет расширить рецептивное поле.
    c. **Расширенная свертка увеличивает рецептивное поле**.
12. Выберите неверное утверждение касательно особенностей двухмерной свертки:
    a. Пространственно-разряженная (разделенная) свертка — это последовательное соединение горизонтального и вертикального прямоугольных ядер.
    b. **Групповая свертка позволяет расширить рецептивное поле.**
    c. Расширенная свертка увеличивает рецептивное поле.

13. Выберите неверное утверждение касательно особенностей двухмерной свертки:
    a. Точечная свертка часто применяется для изменения числа карт признаков.
    b. Глубокая свертка позволяет снизить число параметров слоя.
    c. **Пространственно-разделенная свертка используется для замены одного ядра большой размерности на несколько ядер меньшей размерности.**
14. Выберите верное утверждение касательно особенностей двухмерной свертки:
    a. Пространственно-разделенная свертка часто применяется для изменения числа карт признаков.
    b. Точечная свертка позволяет снизить число параметров слоя.
    c. **Каскадная свертка используется для замены одного ядра большой размерности на несколько ядер меньшей размерности.**
15. Выберите неверное утверждение касательно особенностей двухмерной свертки:
    a. **Пространственно-разделенная свертка часто применяется для изменения числа карт признаков.**
    b. Глубокая свертка позволяет снизить число параметров слоя.

c. Каскадная свертка используется для замены одного ядра большой размерности на несколько ядер меньшей размерности.

16. Выберите верное утверждение касательно особенностей двухмерной свертки:
    d. Точечная свертка часто применяется для изменения числа карт признаков.
    e. Глубокая свертка это замена квадратной свертки на последовательность двух прямоугольных сверток.
    f. Пространственно-разделенная свертка используется для замены одного ядра большой размерности на несколько ядер меньшей размерности.

17. Выберите верное утверждение касательно особенностей слоя глобального пулинга:
    a. Глобальный макс-пулинг наиболее популярная на сегодня реализация идеи данного слоя.
    b. Глобальный пулинг призван решить проблему избыточного числа параметров полносвязного слоя.
    c. Глобальный пулинг призван снизить число карт признаков.
18. Выберите неверное утверждение касательно особенностей слоя глобального пулинга:
    a. Глобальный усредняющий пулинг наиболее популярная на сегодня реализация идеи данного слоя.
    b. Глобальный пулинг призван решить проблему избыточного числа параметров полносвязного слоя.
    c. Глобальный пулинг призван снизить число карт признаков.
19. Выберите верное утверждение касательно особенностей слоя глобального пулинга:
    a. Глобальный макс-пулинг наиболее популярная на сегодня реализация идеи данного слоя.
    b. Глобальный усредняющий пулинг наиболее популярная на сегодня реализация идеи данного слоя.
    c. Глобальный soatmax-пулинг наиболее популярная на сегодня реализация идеи данного слоя.
20. Выберите неверное утверждение касательно особенностей слоя глобального пулинга:
    a. Глобальный усредняющий пулинг наиболее популярная на сегодня реализация идеи данного слоя.
    b. Глобальный пулинг призван решить проблему избыточного числа параметров полносвязного слоя.
    c. Глобальный softmax-пулинг наиболее популярная на сегодня реализация идеи данного слоя.

21. Выберите <u>неверное</u> утверждение касательно особенностей функции активации ReLU:
    a. Функция ReLU иногда вызывает проблемы вымывания градиента.
    b. Функция ReLU иногда имеет проблемы с отсутствием насыщения в области значений больше нуля.
    c. **Функция ReLU имеет проблемы в связи с областью насыщения в производной.**
22. Выберите <u>верное</u> утверждение касательно особенностей функции активации ReLU:
    a. **Функция ReLU иногда вызывает проблемы вымывания градиента.**
    b. Функция ReLU имеет проблемы в связи с несимметричностью.
    c. Функция ReLU имеет проблемы в связи с областью насыщения в производной.
23. Выберите <u>неверное</u> утверждение касательно особенностей функции активации ReLU:
    a. Функция ReLU иногда вызывает проблемы вымывания градиента.
    b. Функция ReLU иногда имеет проблемы с отсутствием насыщения в области значений больше нуля.
    c. **Функция ReLU иногда имеет проблемы в связи с несимметричностью.**
24. Выберите <u>верное</u> утверждение касательно особенностей функции активации ReLU:
    a. **Функция ReLU иногда вызывает проблемы вымывания градиента.**
    b. Функция ReLU имеет проблемы в связи с несимметричностью.
    c. Функция ReLU имеет проблемы в связи отсутствием насыщения в области меньше нуля.

25. Выберите <u>верное</u> утверждение касательно особенностей инициализации весовых параметров:
    a. Наилучшие результаты обучения могут быть достигнуты, в случае, когда весовые параметры инициализированы небольшими равномерно распределенными значениями.
    b. **Наилучшие результаты обучения могут быть достигнуты, в случае, когда весовые параметры инициализированы распределением с дисперсией обратно пропорциональной размеру слоя.**
    c. Наилучшие результаты обучения могут быть достигнуты, в случае, когда весовые параметры инициализированы распределением с постоянной дисперсией.
26. Выберите <u>верное</u> утверждение касательно особенностей инициализации весовых параметров:

a. Наилучшие результаты обучения могут быть достигнуты, в случае, когда весовые параметры инициализированы постоянными значениями.
b. Наилучшие результаты обучения могут быть достигнуты, в случае, когда весовые параметры инициализированы распределением с дисперсией обратно пропорциональной размеру слоя.
c. Наилучшие результаты обучения могут быть достигнуты, в случае, когда весовые параметры инициализированы распределением с постоянной дисперсией.

27. Выберите <u>неверное</u> утверждение касательно особенностей инициализации весовых параметров:
    a. Наилучшие результаты обучения могут быть достигнуты, в случае, когда весовые параметры инициализированы небольшими равномерно распределенными значениями.
    b. Наилучшие результаты обучения могут быть достигнуты, в случае, когда весовые параметры инициализированы распределением с дисперсией обратно пропорциональной размеру слоя.
    c. Наилучшие результаты обучения могут быть достигнуты, в случае, когда все весовые параметры инициализированы разными значениями, а не одинаковыми.
28. Выберите <u>неверное</u> утверждение касательно особенностей инициализации весовых параметров:
    a. Наилучшие результаты обучения могут быть достигнуты, в случае, когда весовые параметры инициализированы распределением с одинаковой дисперсией.
    b. Наилучшие результаты обучения могут быть достигнуты, в случае, когда весовые параметры инициализированы распределением с дисперсией обратно пропорциональной размеру слоя.
    c. Наилучшие результаты обучения могут быть достигнуты, в случае, когда все весовые параметры инициализированы разными значениями, а не одинаковыми.

29. Выберите <u>верное</u> определение функции потерь:
    a. Функция потерь – это метод оценки того, как обучаемая модель подходит для решения поставленной задачи.
    b. Функция потерь показывает точность работы модели для решаемой задачи.
    c. Функция потерь позволяет оценить, например, число правильных ответов среди всех или другой схожий показатель среднего качества работы модели.
30. Выберите <u>верное</u> определение функции потерь:
    a. Функция потерь – это метод оценки того, как обучаемая модель подходит для решения поставленной задачи.

b. Функция потерь показывает отношение числа правильных ответов для одного класса к числу всех ответов для этого класса.
c. Функция потерь показывает отношение числа правильных ответов для одного класса к числу правильных ответов для всех классов.

31. Выберите верное определение функции потерь:
    a. **Функция потерь – это метод оценки того, как обучаемая модель подходит для решения поставленной задачи.**
    b. Функция потерь показывает отношение числа неправильных ответов для одного класса к общему числу неправильных ответов.
    c. Функция потерь позволяет оценить, например, число правильных ответов среди всех или другой схожий показатель среднего качества работы модели.
32. Выберите верное определение функции потерь:
    a. **Функция потерь – это метод оценки того, как обучаемая модель подходит для решения поставленной задачи.**
    b. Функция потерь показывает точность работы модели для решаемой задачи.
    c. Функция потерь показывает полноту работы модели для решаемой задачи.

33. Выберите неверный вариант функции потерь для решения задачи семантической сегментации:
    a. **Межканальная среднеквадратичная ошибка (по пикселям с одной пространственной позицией).**
    b. Межканальная категориальная кросс-энтропия.
    c. Функция (коэффициент) Дайс.
34. Выберите верный вариант функции потерь для решения задачи семантической сегментации:
    a. Межканальная среднеквадратичная ошибка (по пикселям с одной пространственной позицией).
    b. **Межканальная категориальная кросс-энтропия + Функция (коэффициент) Дайс.**
    c. Среднее разности пикселей полученного и целевого результата.
35. Выберите неверный вариант функции потерь для решения задачи семантической сегментации:
    a. **Среднее разности пикселей полученного и целевого результата.**
    b. Межканальная категориальная кросс-энтропия.
    c. Функция (коэффициент) Дайс.
36. Выберите верный вариант функции потерь для решения задачи семантической сегментации:
    a. Среднее разности пикселей полученного и целевого результата.
    b. Межканальная среднеквадратичная ошибка (по пикселям с одной пространственной позицией).
    c. **Функция (коэффициент) Дайс.**

37. Выберите <u>верный</u> <u>вариант</u> причины использования регуляризации:
    a. **Снижение проблемы неустойчивости результатов обучения при введении смещения в данные.**
    b. Снижение времени обучения.
    c. Повышение точности обучения для тренировочной выборки.
38. Выберите <u>верный</u> <u>вариант</u> причины использования регуляризации:
    a. **Снижение проблемы неустойчивости результатов обучения при введении смещения в данные.**
    b. Снижение требований к выбору параметров обучения.
    c. Повышение точности обучения для тренировочной выборки.
39. Выберите <u>неверный</u> <u>вариант</u> причины использования регуляризации:
    a. Снижение проблемы неустойчивости результатов обучения при введении смещения результатов.
    b. Повышение обобщающей способности.
    c. **Повышение точности обучения для тренировочной выборки.**
40. Выберите <u>неверный</u> <u>вариант</u> причины использования регуляризации:
    a. Снижение проблемы неустойчивости результатов обучения при введении смещения результатов.
    b. Повышение обобщающей способности.
    c. Снижение времени обучения.

41. Выберите <u>верный</u> вариант причины использования метод дрпоаут:
    a. **Снижение вероятности возникновения проблемы соадоптации.**
    b. Снижение требований к выбору скорости обучения и значениям других гиперпараметров.
    c. Снижение вероятности возникновения проблемы взрыва градиента.
42. Выберите <u>верный</u> вариант причины использования метод дрпоаут:
    a. **Снижение вероятности возникновения проблемы соадоптации.**
    b. Снижение требований к выбору скорости обучения и значениям других гиперпараметров.
    c. Увеличение скорости обучения.
43. Выберите <u>верный</u> вариант причины использования метод дрпоаут:
    a. **Снижение вероятности возникновения проблемы соадоптации.**
    b. Снижение вероятности возникновения проблемы вымывания градиента.
    c. Снижение вероятности возникновения проблемы взрыва градиента.
44. Выберите <u>верный</u> вариант причины использования метод дрпоаут:
    a. **Снижение вероятности возникновения проблемы соадоптации.**
    b. Увеличение скорости обучения.

c. Снижение вероятности возникновения проблемы вымывания градиента.

45. Выберите <u>верный</u> вариант причины использования метод батч нормализации:
    a. Снижение вероятности возникновения проблемы соадоптации.
    b. Снижение вероятности возникновения проблемы ковариационного сдвига или других проблем разброса значений.
    c. Снижение требований к выбору размера батча.
46. Выберите <u>неверный</u> вариант причины использования метод батч нормализации:
    a. Ускорение обучения.
    b. Снижение вероятности возникновения проблемы ковариационного сдвига или других проблем разброса значений.
    c. Снижение требований к выбору размера батча.
47. Выберите <u>верный</u> вариант причины использования метод батч нормализации:
    a. Снижение вероятности возникновения проблемы соадоптации.
    b. Снижение вероятности возникновения проблемы ковариационного сдвига или других проблем разброса значений.
    c. Повышение устойчивости при переносе обучения.
48. Выберите <u>неверный</u> вариант причины использования метод батч нормализации:
    a. Снижение вероятности возникновения проблемы соадоптации.
    b. Снижение вероятности возникновения проблемы ковариационного сдвига или других проблем разброса значений.
    c. Ускорение обучения.

49. Выберите <u>верный</u> вариант недостатка метода батч нормализации:
    a. Снижение точности в случае небольшого или переменного размера батча.
    b. Требования более тщательного выбора значения скорости обучения или других параметров.
    c. Повышение вероятности возникновения проблемы вымывания градиента.
50. Выберите <u>верный</u> вариант недостатка метода батч нормализации:
    a. Снижение точности в случае небольшого или переменного размера батча.
    b. Повышение вероятности возникновения проблемы взрыва градиента.
    c. Повышение вероятности возникновения проблемы вымывания градиента.
51. Выберите <u>верный</u> вариант недостатка метода батч нормализации:

a. **Снижение точности в случае небольшого или переменного размера батча.**
b. Требования более тщательного выбора значения скорости обучения или других параметров.
c. Повышение вероятности возникновения проблемы соадоптации.

52. Выберите верный вариант недостатка метода батч нормализации:
a. **Снижение точности в случае небольшого или переменного размера батча.**
b. Повышение вероятности возникновения проблемы соадоптации.
c. Повышение вероятности возникновения проблемы взрыва градиента.

53. Выберите неверный вариант, касающийся особенностей различных методов нормализации:
a. Слой LayerNorm работает одинаково как при тренировке, так и при тестировании.
b. **Слой GroupNorm предназначен только для батчей большого размера.**
c. В случае небольшого размера батча часто рекомендуется использовать нормализацию (или стандартизацию) весов.

54. Выберите верный вариант, касающийся особенностей различных методов нормализации:
a. **Слой LayerNorm работает одинаково как при тренировке, так и при тестировании.**
b. Слой GroupNorm предназначен только для батчей большого размера.
c. В случае небольшого размера батча рекомендуется использовать слой BatchNorm.

55. Выберите верный вариант, касающийся особенностей различных методов нормализации:
a. Слой BatchNorm работает одинаково как при тренировке, так и при тестировании.
b. Слой GroupNorm предназначен только для батчей большого размера.
c. **В случае небольшого размера батча часто рекомендуется использовать нормализацию (или стандартизацию) весов.**

56. Выберите неверный вариант, касающийся особенностей различных методов нормализации:
a. Слой BatchNorm работает по-разному при тренировке, и при тестировании.
b. **Слой GroupNorm предназначен только для батчей большого размера**.
c. В случае небольшого размера батча не рекомендуется использовать слой BatchNorm.

57. Выберите неверный вариант касающийся метода кросс валидации:

a. Метод кросс валидации Hold-Out Cross-Validation наиболее общий выбор.
b. Метод k-Fold Cross-Validation может быть использован для выбора наилучшей модели.
c. **Метод Hold-Out Cross-Validation следует использовать для несбалансированных данных.**

58. Выберите <u>верный</u> вариант касающийся метода кросс валидации:
    a. Метод кросс валидации LOO Cross-Validation наиболее общий выбор.
    b. **Метод k-Fold Cross-Validation может быть использован для выбора наилучшей модели.**
    c. Метод Hold-Out Cross-Validation следует использовать для несбалансированных данных.
59. Выберите <u>верный</u> вариант касающийся метода кросс валидации:
    a. Метод кросс валидации k-Fold Cross-Validation наиболее общий выбор.
    b. **Метод k-Fold Cross-Validation может быть использован для выбора наилучшей модели.**
    c. Метод Hold-Out Cross-Validation следует использовать для несбалансированных данных.
60. Выберите <u>неверный</u> вариант касающийся метода кросс валидации:
    a. Метод кросс валидации Hold-Out Cross-Validation наиболее общий выбор.
    b. Метод LOO Cross-Validation может быть использован для выбора наилучшей модели.
    c. **Метод Hold-Out Cross-Validation следует использовать для несбалансированных данных.**

61. Выберите <u>верный</u> вариант утверждения касательно Стохастического градиентного спуска (SGD):
    a. **Метод SGD рекомендуется использовать с моментом, особенно для небольших размеров батча.**
    b. Разбиение на батчи лучше проводить единожды и перед началом процедуры тренировки.
    c. Использование переменной скорости обучения необходимо только для подбора ее правильного значения в SGD – то есть в качестве меры предварительного обучения.
62. Выберите <u>верный</u> вариант утверждения касательно Стохастического градиентного спуска (SGD):
    a. **Метод SGD рекомендуется использовать с моментом, особенно для небольших размеров батча.**
    b. Рекомендуется использовать SGD второго порядка, так они сходятся быстрее.
    c. Использование переменной скорости обучения необходимо только для подбора ее правильного значения в SGD – то есть в качестве меры предварительного обучения.
63. Выберите <u>неверный</u> вариант утверждения касательно Стохастического градиентного спуска (SGD):

a. Метод SGD рекомендуется использовать с моментом, особенно для небольших размеров батча.
b. Разбиение на батчи лучше проводить динамически в ходе процедуры тренировки.
c. **Использование переменной скорости обучения необходимо только для подбора ее правильного значения в SGD – то есть в качестве меры предварительного обучения.**

64. Выберите <u>неверный</u> вариант утверждения касательно Стохастического градиентного спуска (SGD):
    a. Метод SGD рекомендуется использовать с моментом, особенно для небольших размеров батча.
    b. **Разбиение на батчи лучше проводить единожды и перед началом процедуры тренировки.**
    c. Использование переменной скорости обучения предпочтительно для снижения вероятности попадания в локальные минимумы рельефа функции потерь.

65. Выберите <u>верный</u> вариант утверждения касательно адаптивных методов стохастического градиентного спуска:
    a. Метод RMSProp не включает момент.
    b. **Методы адаптивного спуска не нуждаются в выборе переменной скорости обучения.**
    c. Метод ADAM не включает момент.
66. Выберите <u>неверный</u> вариант утверждения касательно адаптивных методов стохастического градиентного спуска:
    a. **Метод RMSProp не включает момент**.
    b. Методы адаптивного спуска нуждаются в выборе переменной скорости обучения.
    c. Метод ADAM включает момент.
67. Выберите <u>верный</u> вариант утверждения касательно адаптивных методов стохастического градиентного спуска:
    a. Метод RMSProp рекомендуется с использованием переменной скорости обучения.
    b. **Методы SGD рекомендуется с использованием переменной скорости обучения.**
    c. Метод ADAM рекомендуется с использованием переменной скорости обучения.
68. Выберите <u>неверный</u> вариант утверждения касательно адаптивных методов стохастического градиентного спуска:
    a. Метод RMSProp не рекомендуется использовать с переменной скорости обучения.
    b. **Методы SGD не рекомендуется использовать с переменной скорости обучения.**
    c. Метод ADAM не рекомендуется использовать с переменной скорости обучения.

69. Выберите <u>верный</u> вариант утверждения касательно архитектуры VGG:

a. **Особенность архитектуры VGG – использование каскадной свертки.**
b. Классические реализации архитектур VGG имеют число параметров меньше, чем AlexNet.
c. В основе архитектуры VGG структура архитектуры LeNet.

70. Выберите верный вариант утверждения касательно архитектуры VGG:
    d. **Особенность архитектуры VGG – использование каскадной свертки.**
    e. Особенность архитектуры VGG – использование групповой свертки.
    f. Особенность архитектуры VGG – использование пространственно-разряженной (разделенной) свертки.
71. Выберите неверный вариант утверждения касательно архитектуры VGG:
    a. Особенность архитектуры VGG – использование каскадной свертки.
    b. Архитектура VGG одна из самых больших по соотношению число параметров к числу слоёв.
    c. **Особенность архитектуры VGG – использование пространственно-разряженной (разделенной) свертки.**
72. Выберите неверный вариант утверждения касательно архитектуры VGG:
    a. Особенность архитектуры VGG – использование каскадной свертки.
    b. Архитектура VGG одна из самых больших по соотношению число параметров к числу слоёв.
    c. **В основе архитектуры VGG структура архитектуры LeNet.**

73. Выберите неверный вариант утверждения касательно архитектуры NiN:
    a. Архитектура InceptionNet (GoogLeNet) это вариант развития идей NiN.
    b. В основе подхода NiN лежит идея обучения нескольких нейронных сетей и использование одной дополнительной сети, обученной по результатам предыдущих.
    c. **Предполагается, что за счет разветвления градиента в NiN разные части слоя могут выделять различные признаки.**
74. Выберите верный вариант утверждения касательно архитектуры NiN:
    a. **Архитектура InceptionNet (GoogLeNet) это вариант развития идей NiN.**
    b. Архитектура ResNet это вариант развития идей NiN.
    c. Предполагается, что за счет разветвления градиента в NiN разные части слоя могут выделять различные признаки.
75. Выберите неверный вариант утверждения касательно архитектуры NiN:
    a. Архитектура InceptionNet (GoogLeNet) это вариант развития идей NiN.
    b. В основе подхода NiN лежит идея обучения нескольких нейронных сетей и использование одной дополнительной сети, обученной по результатам предыдущих.
    c. **Архитектура ResNet это вариант развития идей NiN.**
76. Выберите верный вариант утверждения касательно архитектуры NiN:

a. **Архитектура InceptionNet (GoogLeNet) это вариант развития идей NiN.**
b. Архитектура ResNet это вариант развития идей NiN.
c. Архитектура Transformer это вариант развития идей NiN.

77. Выберите неверный вариант утверждения касательно обоснования работоспособности архитектур ResNet.
    a. **Остаточный слой снижает требования к размеру набора данных, так как позволяет проводить регуляризацию.**
    b. Остаточный слой снижает вероятность возникновения переобучения, так как позволяет проводить регуляризацию остаточными связями.
    c. Остаточный слой позволяет наращивать глубину сети за счет остаточных связей.
78. Выберите верный вариант утверждения касательно обоснования работоспособности архитектур ResNet.
    a. Остаточный слой снижает проблему соадоптации.
    b. Остаточный слой снижает проблему ковариационного сдвига.
    c. **Остаточный слой позволяет наращивать глубину сети за счет остаточных связей.**
79. Выберите неверный вариант утверждения касательно обоснования работоспособности архитектур ResNet.
    a. **Остаточный слой снижает проблему ковариационного сдвига.**
    b. Остаточный слой снижает вероятность возникновения переобучения так как позволяет проводить регуляризацию остаточными связями.
    c. Остаточный слой позволяет наращивать глубину сети за счет остаточных связей.
80. Выберите верный вариант утверждения касательно обоснования работоспособности архитектур ResNet.
    a. Остаточный слой снижает проблему соадоптации.
    b. **Остаточный слой снижает вероятность возникновения переобучения так как позволяет проводить регуляризацию остаточными связями.**
    c. Остаточный слой снижает проблему ковариационного сдвига.

81. Выберите неверный вариант утверждения касательно обоснования работоспособности архитектур ResNet.
    a. Необходимо использовать одинаковый размер карт признаков на входе и выходе блока с остаточными связями.
    b. Если число карт признаков на входе и выходе блока с остаточными связями разное необходимо использовать точечную свертку.
    c. **Рекомендуется использовать слой дропаута в составе блока ResNet.**
82. Выберите неверный вариант утверждения касательно обоснования работоспособности архитектур ResNet.

   a. Если число карт признаков на входе и выходе блока с остаточными связями разное необходимо использовать точечную свертку.
   b. Желательно использовать одинаковое число карт признаков на входе и выходе блока с остаточными связями.
   c. Рекомендуется использовать слой дропаута в составе блока ResNet.

83. Выберите <u>неверный</u> вариант утверждения касательно обоснования работоспособности архитектур ResNet.
   a. Блок ResNet не подразумевает слои пулинга.
   b. Если число карт признаков на входе и выходе блока с остаточными связями разное необходимо использовать точечную свертку.
   c. Рекомендуется использовать слой дропаута в составе блока ResNet.
84. Выберите <u>неверный</u> вариант утверждения касательно обоснования работоспособности архитектур ResNet.
   a. Необходимо использовать одинаковый размер карт признаков на входе и выходе блока с остаточными связями.
   b. Блок ResNet не подразумевает слои пулинга.
   c. Рекомендуется использовать слой дропаута в составе блока ResNet.

85. Выберите <u>неверный</u> вариант утверждения касательно особенностей архитектур DenseNet.
   a. Блок DenseNet позволяет принимать во внимание низко размерные детали изображений за счет набора остаточных связей.
   b. Число параметров архитектуры DenseNet как правило выше, чем для ResNet.
   c. Блок DenseNet может иметь разное число карт признаков на входе и на выходе.
86. Выберите <u>верный</u> вариант утверждения касательно особенностей архитектур DenseNet.
   a. Рекомендуется использовать слой дропаута в составе блока DenseNet.
   b. Число параметров архитектуры DenseNet как правило выше, чем для ResNet.
   c. Блок DenseNet может иметь разное число карт признаков на входе и на выходе.
87. Выберите <u>неверный</u> вариант утверждения касательно особенностей архитектур DenseNet.
   a. Блок DenseNet позволяет принимать во внимание низко размерные детали изображений за счет набора остаточных связей.
   b. Рекомендуется использовать слой дропаута в составе блока DenseNet.
   c. Блок DenseNet может иметь разное число карт признаков на входе и на выходе.

88. Выберите <u>неверный</u> вариант утверждения касательно особенностей архитектур DenseNet.
    a. Рекомендуется использовать слой дропаута в составе блока DenseNet.
    b. Блок DenseNet может иметь разное число карт признаков на входе и на выходе.
    c. Число параметров архитектуры DenseNet как правило выше, чем для ResNet.

89. Выберите <u>неверный</u> вариант утверждения касательно особенностей архитектур MobileNet
    a. Блок MobileNet включает слой расширения и слой проекции, где степень расширение – это гиперпараметр архитектуры.
    b. Блок MobileNet использует DeepWise-Separable свертку.
    c. Блок MobileNet не использует остаточные связи – сеть и так небольшая.
90. Выберите <u>неверный</u> вариант утверждения касательно особенностей архитектур MobileNet
    a. Блок MobileNet включает слой расширения и слой проекции, где степень расширение – это гиперпараметр архитектуры.
    b. Блок MobileNet использует DeepWise-Separable свертку.
    c. Блок MobileNet использует реугляризацию методом Dropout.
91. Выберите <u>верный</u> вариант утверждения касательно особенностей архитектур MobileNet
    a. Блок MobileNet включает слой расширения и слой проекции, где степень расширение – это гиперпараметр архитектуры.
    b. Блок MobileNet использует реугляризацию методом Dropout.
    c. Блок MobileNet не использует остаточные связи – сеть и так небольшая.
92. Выберите <u>верный</u> вариант утверждения касательно особенностей архитектур MobileNet
    a. Блок MobileNet использует реугляризацию методом Dropout.
    b. Блок MobileNet использует DeepWise-Separable свертку.
    c. Блок MobileNet не использует остаточные связи – сеть и так небольшая.

93. Выберите <u>неверный</u> вариант утверждения касательно особенностей архитектур блока Squeeze-and-Excitation, (SE):
    a. Блок SE позволяет подсветить наиболее важные признаками.
    b. Блок SE сжимает пространственные размерности карт признаков.
    c. Блок SE имеет степень расширения как гиперпараметр.
94. Выберите <u>неверный</u> вариант утверждения касательно особенностей архитектур блока Squeeze-and-Excitation, (SE):
    a. Блок SE позволяет подсветить наиболее важные признаками.
    b. Блок SE сжимает пространственные размерности карт признаков.
    c. Блок SE имеет глубину как гиперпараметр.

95. Выберите верный вариант утверждения касательно особенностей архитектур блока Squeeze-and-Excitation, (SE):
    a. Блок SE имеет глубину как гиперпараметр.
    b. Блок SE сжимает пространственные размерности карт признаков.
    c. Блок SE имеет степень расширения как гиперпараметр.
96. Выберите верный вариант утверждения касательно особенностей архитектур блока Squeeze-and-Excitation, (SE):
    a. Блок SE позволяет подсветить наиболее важные признаками.
    b. Блок SE имеет глубину как гиперпараметр.
    c. Блок SE имеет степень расширения как гиперпараметр.

97. Выберите неверный вариант утверждения касательно особенностей архитектуры Efficient Net:
    a. Efficient Net получена методом автоматического поиска архитектур.
    b. Efficient Net использует блоки типа MobileNet.
    c. Семейство архитектур Efficient Net предназначены для работы на мобильных и портативных устройствах.
98. Выберите неверный вариант утверждения касательно особенностей архитектуры Efficient Net:
    a. Efficient Net получена методом автоматического поиска архитектур.
    b. Efficient Net использует блоки типа Inception.
    c. Efficient Net имеет ряд вариантов, полученных методом compound scaling.
99. Выберите неверный вариант утверждения касательно особенностей архитектуры Efficient Net:
    a. Efficient Net получена методом автоматического поиска архитектур.
    b. Efficient Net имеет ряд вариантов, полученных методом compound scaling.
    c. Семейство архитектур Efficient Net предназначены для работы на мобильных и портативных устройствах.
100. Выберите верный вариант утверждения касательно особенностей архитектуры Efficient Net:
    a. Efficient Net использует блоки типа Inception.
    b. Efficient Net использует блоки типа MobileNet.
    c. Efficient Net использует блоки типа shuffle net.